Reconfigurable Hardware in Wearable Computing Nodes

Christian Plessl, Rolf Enzler, Herbert Walder, Jan Beutel, Marco Platzner, Lothar Thiele
Swiss Federal Institute of Technology (ETH) Zurich,* Switzerland
plessl@tik.ee.ethz.ch

Abstract

Wearable computers are embedded into the mobile environment of the human body. A design challenge for wearable systems is to combine the high performance required for tasks such as video decoding with low energy consumption required to maximize 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 two experiments that show the uses of reconfigurable hardware in WURM: ASICs-on-demand and adaptive interfaces. Finally, we develop and evaluate task placement techniques used in the operating system layer of WURM.

1 Introduction

Many current wearable systems are built out of standard components such as personal digital assistants or sub-notebooks. These more or less miniaturized general-purpose computers are very attractive due to their easy availability, low price, and matured development tools including compilers and operating systems.

In this paper, we argue that future wearable computing systems should be viewed and designed as *embedded systems*. A wearable computer is embedded into a mobile environment, the human body, and reacts to events from this environment. We denote such a computing system as *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 centered around one general-purpose main module.

Sensors and actors are distributed over the human body. The current efforts in integration of electronics and wires into clothing, shoes and other appliances promote this distribution. Examples are acceleration sensors integrated in garments or kinetic energy converters in shoes. The distribution of sensors and actors implies that control and preprocessing is bound to specific locations on the body. For instance, the preprocessing of sampled audio data, i.e., filtering and compression, is best done directly at the microphone node.

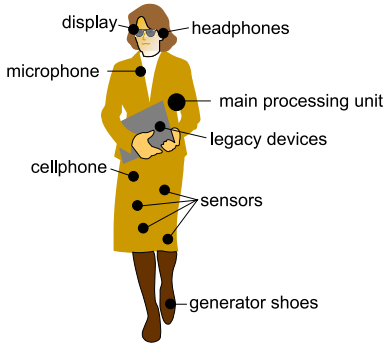

Figure 1. The body area computing system.

The communication network is a mixture of wireless and wired connections. Presently, wireless is the predominant technology. In the future, conducting textiles might increase the trend toward wired connections for nodes in the same piece of textile. External networks 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:

- **Multi-mode performance:** Wearable systems are multi-mode systems. They require a fixed baseline amount of performance for running control tasks that do neither show high computational demands nor stringent timing constraints. Occasionally, wearable systems execute bursts of computation-intensive tasks that carry real-time constraints [1]. An example for a task that can be served best effort is reading position sensors to update the geographical context. Examples for real-time tasks are audio and video coding. Wearable

*This work is carried out in cooperation with the Wearable Computing Laboratory at ETH Zurich and is partially supported by ETH Zurich and the Swiss National Science Foundation (SNF).

Published in: *Proc. Int. Symp. on Wearable Computers (ISWC)*
 DOI: 10.1109/ISWC.2002.1167250

computing systems must have sufficient performance to satisfy the real-time constraints. Missed deadlines render the system useless.

- **Low energy:** The energy awareness of the body area computing system is essential since we want the wearable system on and functional the whole day. Energy awareness comprises several measures. First, we have to run the high performance bursts on computing elements with high energy efficiency. Second, to provide the baseline performance computing elements optimized for low-power must be used. Third, a dynamic power manager should force the components into their power-down modes or even shut them off, depending on the high-level context.
- **High flexibility:** The body area computing system has to handle highly dynamic situations. First, the application requirements vary strongly with the user's choices, but also with the context and the location. Second, as clothes are put on and off, components are dynamically added and removed from the body area computing system. On a longer time-scale, wearable systems have to adapt to emerging or changing communication standards and protocols.

Further design criteria include 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.

On one hand, the flexibility requirements in wearable computing demand a programmable general-purpose computing system. On the other hand, the high performance and low energy requirements ask for a specialized computing system. To answer this design challenge, we propose to include *reconfigurable hardware* in the computing nodes of the body area computer system. A similar concept has been presented by Yau and Karim [2]. They discuss a reconfigurable middleware layer for distributed systems that implements the performance-intensive parts in FPGAs. In contrast to that, we do not restrict reconfigurable hardware to the middleware layer. In this paper, we present wearable nodes comprised of small- to medium-range low-power 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.

Section 2 discusses the advantages of reconfigurable hardware in body area computing systems. Section 3 introduces the concept of a Wearable Unit with Reconfigurable Modules (WURM). In Section 4, we present a WURM prototype implementation and report on several experiments with the prototype. Section 5 outlines further work.

2 Advantages of Reconfigurable Hardware in Wearable Computing

In this section, we first review the main characteristics of reconfigurable hardware compared to processors and dedicated hardware and then advocate its use in wearable computing nodes.

2.1 The Efficiency of Reconfigurable Hardware

Most reconfigurable elements base on field-programmable gate array (FPGA) technology. FPGAs consist of an array of programmable logic blocks which are interconnected by programmable routing channels. SRAM cells are used to control the functionality of the logic blocks and the routing. SRAM-based FPGAs 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 logic capacity, today's devices feature millions of gates of programmable logic.

With respect to performance, power consumption and flexibility, reconfigurable hardware is positioned between processors and dedicated hardware. For a given function, dedicated hardware achieves the highest performance and the lowest power consumption due to the high degree of specialization. However, dedicated hardware is also totally 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. Several case studies have shown that FPGAs achieve higher throughputs and are more power efficient than processors, provided that the application matches well the spatial structures of FPGAs and reveals a sufficient amount of parallelism.

Mencer et al. [3] compared different implementations of the IDEA cryptography algorithm on RISC and DSP processors and on an FPGA. Abnous et al. [4] performed similar studies on finite and infinite impulse response filters (FIR, IIR). Table 1 summarizes the results and gives throughput and energy efficiency for each application. For IDEA, the throughput is measured in million encrypted bits per second, for FIR and IIR the throughput is measured in million 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.

Recently, hybrid CPUs that integrate FPGAs with CPU

Type	Device	IDEA [3]		FIR [4]		IIR [4]	
		$[Mbit/s]$	$[Mbit/Ws]$	$[Mtap/s]$	$[Mtap/Ws]$	$[Mtap/s]$	$[Mtap/Ws]$
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

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

cores on a System-on-Chip have emerged. Examples are Triscend's 8051-based E5 or Xilinx' PowerPC-based Virtex-II Pro. The tight integration of CPU and FPGA allows for low-latency and low-power communication. Stitt et al. [5] studied the energy efficiency of hybrid CPUs for embedded systems and evaluated a set of benchmarks that are relevant to wearable computing. On the Triscend E5 device, they measured energy savings of 71% on average by moving application kernels to the FPGA instead of running the applications solely on the CPU.

2.2 Reconfigurable Hardware in Wearables

We see two main uses of reconfigurable hardware in a wearable computing node:

- **ASIC-on-demand:** Compute-intensive functions with timing constraints are 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. Especially small wearable nodes with attached sensors benefit greatly from ASICs-on-demand. Such nodes often require compute-intensive preprocessing to reduce data rates. This is motivated by the fact that wireless communication is more expensive than computation in terms of energy [6].

  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, [7] describes a feature extractor that bases 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 execute the preprocessing in reconfigurable hardware.

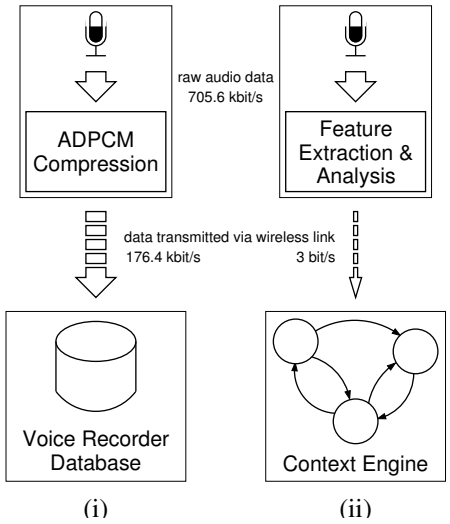

Figure 2. Two audio applications running on the same wearable node: (i) voice recorder, (ii) context recognition.

- **Adaptive interfaces:** For economic reasons, we are interested in re-using 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 [8]. In another scenario, a chest-bound node that discriminates between indoor and outdoor might have only one or two light sensors attached to it.

Adaptive interfaces can be combined with ASICs-on-demand. Reconfigurable hardware can compute interface conversions and protocol stacks to relieve the CPU from such tasks.

3 Wearable Unit with Reconfigurable Modules (WURM)

This section presents the WURM (*W*earable *U*nit with *R*econfigurable *M*odules) architectural concept for a basic node of the body area computing system.

3.1 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 realizations 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 number of reconfigurable logic blocks might be sufficient.

3.2 WURM Software Architecture

The WURM software architecture is shown in Figure 4. 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., CPU, reconfigurable hardware unit, memory, and I/O. The WURM-OS

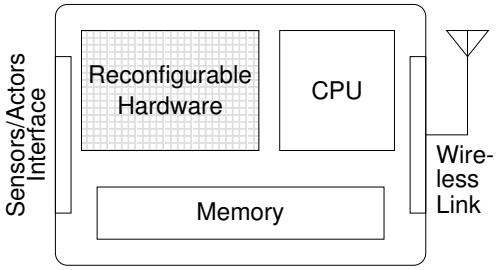

Figure 3. WURM hardware architecture.

layer provides a set of system services to the higher levels of the software architecture. The main service of WURM-OS is the execution of software and hardware tasks. Software tasks run on the node's CPU, hardware tasks execute on the reconfigurable hardware unit. We envision system and application layers on top of WURM-OS. The system layer is part of a distributed operating system that manages the complete body area computing system. Typical system tasks are status checks and the loading of hardware and software tasks. Application tasks define the functional behavior of the node.

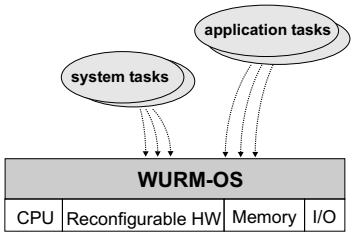

Figure 4. WURM software architecture.

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 a quite new topic and involves several challenges. While the CPU executes tasks quasi-parallel by preemption, 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 tasks and I/O
- scheduling and placement 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. We describe tasks as polyominoes, i.e., as a set of connected rectangular subareas. Such a task shape is the natural result of a core-oriented design style. Figure 8 shows an example of a communication protocol stack that splits up into several rectangular-shaped subtasks. Further structural characteristics are *relocatability* and *transformability*. 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. A transformable task comprises a number of relocatable subtasks.

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

4 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 with a low energy consumption in a flexible way. The WURM prototype is work in progress. In the following sections, we present our prototyping platform and a number of experiments toward ASICs-on-demand and adaptive interfaces. Finally, we discuss task placement in WURM-OS and report on simulation experiments for different placement techniques.

4.1 WURM Prototyping Platform

Figure 5 shows the block diagram of the WURM prototyping platform. The platform bases 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 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 [9] to 32-bit RISCs running at 125 MHz [10]. 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

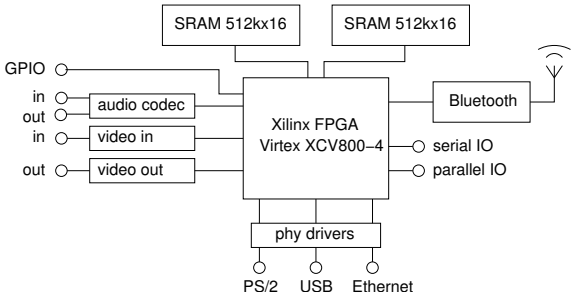

Figure 5. Block diagram of the WURM prototyping platform.

controlled by a host system via the parallel interface. In the future, we plan to switch to one of the upcoming hybrid devices that integrate a dedicated CPU and a reconfigurable array in a System-on-Chip, e.g. Xilinx Virtex-II Pro.

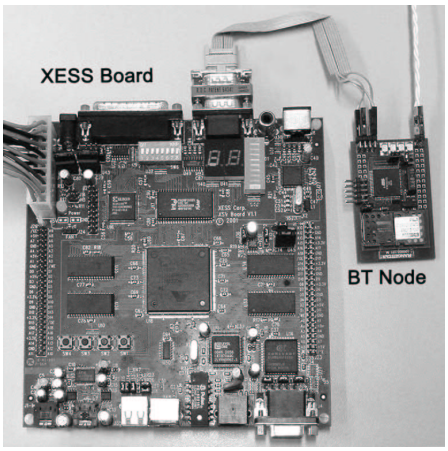

Figure 6. WURM prototyping platform.

For wireless communication between the WURM nodes, we have developed the Bluetooth module BTnode [11]. 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 reconfigurable hardware unit computes runtime-intensive functions more efficiently than the CPU. Runtime-intensive functions are found in applications from the areas multimedia, cryptography, and communication. To identify the portions of code that should go into the reconfigurable

hardware, i.e., the kernels, we have analyzed a set of 29 benchmark programs (MCCmix). We have profiled the programs with a cycle-accurate CPU simulator and have derived execution statistics. Figure 7 shows the relative runtimes of the two most dominating functions for each application. It is notable that on average the MCCmix applications spend 59% of their runtime in their kernels, as opposed to only 19% for the SPEC benchmarks. The detailed results of our analysis have been published elsewhere [12].

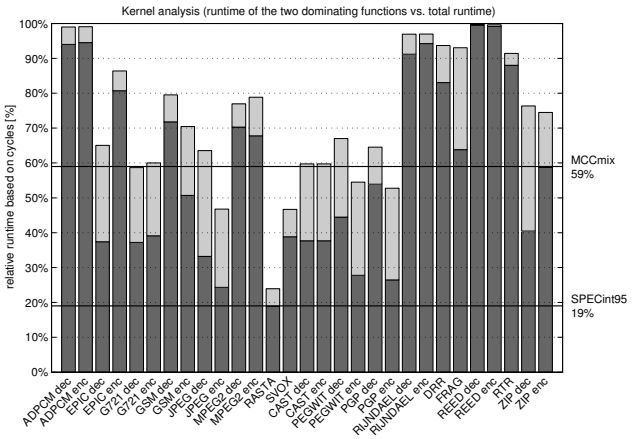

Figure 7. Kernel identification for wearable applications.

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 asks for ASICs-on-demand. We have chosen ADPCM (adaptive differential pulse code modulation) from our benchmark set for a detailed case study [13].

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 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 hardware task, receives and buffers the packets. The UDP protocol is executed on the CPU as software task. The CPU recognizes the used encoding format and initiates the dynamic configuration of a 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 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:

- The soft CPU core used for this experiment is the SPARC V8 compatible 32-bit LEON CPU [14]. The CPU implements 2 kB 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 any optimization, 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 25 MHz clock frequency, this results in a decoding performance of 350 Ksamples/sec. The ADPCM hardware task decodes one sample in 5 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 gives 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 SSI (synchronous serial interface) interfaces to an Ethernet media access controller. Table 2 lists the typical sizes of such tasks, measured in number of Virtex slices.

Hardware Task	Area $[slices]$
UART [15]	100
SSI (PCM serializer) [13]	38
Ethernet-MAC [15]	628
ARP (part of Ethernet-MAC) [15]	417
IP (Internet Protocol) [15]	690
UDP (User Datagram Protocol) [15]	365

Table 2. Area requirements for communication tasks.

To demonstrate the feasibility of protocol stack processing in reconfigurable hardware, we have devised a minimal IP stack as hardware task. The hardware IP stack handles the ARP, IP and UDP protocols autonomously without intervention of the CPU. Figure 8 shows the floorplan of the IP stack in the FPGA.

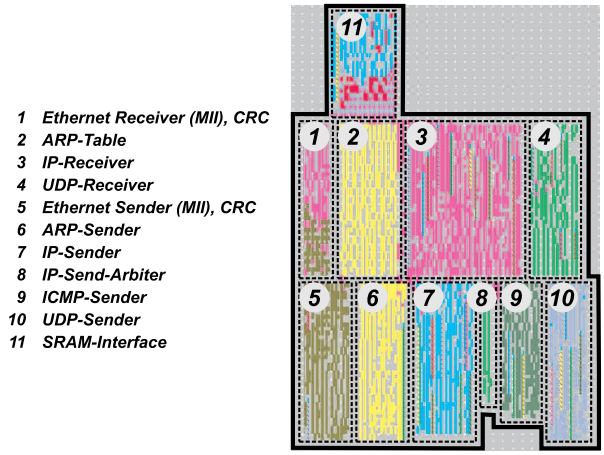

Figure 8. Floorplan of the minimal IP stack.

In order to test the various communication interfaces, we have developed a WURM that implements a reconfigurable Bluetooth/Ethernet bridge [15]. 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 8.

4.4 Task Placement in WURM-OS

The placement of ready tasks into the reconfigurable hardware is a central function of WURM-OS. A clever placement packs tasks tightly in order to minimize the wasted space. Figure 9 sketches an allocation of four tasks T_1-T_4 in the reconfigurable hardware array and a ready task T_n to be placed. Tight task packing allows for the placement and execution of more tasks in parallel, which increases the node's performance.

Online task placement on partially reconfigurable FPGAs is related to 2D bin-packing. Therefore, algorithms used in 2D bin-packing, such as first-fit, best-fit, or bottom-left heuristics, have been considered for task placement [16]. All this approaches assume rectangular task shapes. Since in WURM-OS tasks are polyominoes rather than rectangles, we have devised variants of first-fit and best-fit. Additionally, we have experimented with footprint transforms that change the task's shapes to fit them into the available area. Given a new ready task, the different placement techniques work as follows:

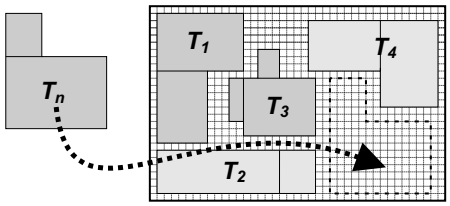

Figure 9. Placement of a ready task T_n into the reconfigurable array.

- *First-fit* searches the reconfigurable array row-wise from the bottom-left to the top-right corner and tries to match the task shape with free blocks. The task is mapped to the first matching position found.

- *First-fit + Footprint transform* is basically a first-fit technique. When no direct fit for a task can be found, the shape of the task is modified by translating its subtasks to different positions. Footprint transforms are done with the restriction that the overall task area must remain contiguous.

- *Best-fit* places a task in a way that minimizes the fragmentation of the residual area [16]. To quantify allocation situations we have introduced the fragmentation grade F. For a given allocation, best-fit places the largest possible rectangle into the residual area, notes its dimensions and marks it as considered. This process is iteratively continued with the next-largest rectangle, until the complete free area has been marked. The result is a histogram of free rectangular areas. The histogram consists of i classes, each denoting n_i rectangles of size a_i. The fragmentation grade is then given as:

$$F = 1 - \frac{\sqrt{\sum_i (n_i \cdot a_i^2)}}{\sum_i (n_i \cdot a_i)}$$

  Best-fit checks all possible placements of a task and selects the position that minimizes the resulting fragmentation grade.

To evaluate the placement qualities of this techniques, we have conducted simulations with randomly generated task sets [17]. We have simulated the placement of a set of hardware tasks on a reconfigurable array of limited size and have measured the overall execution time. We assume that hardware tasks cannot be preempted. We have generated five classes of task sets that differ in task sizes but have equal distributions in task shapes, arrival times, and execution times. The classes are denoted by C_j and contain tasks with sizes equally distributed in the interval $[100, j]$ logic blocks.

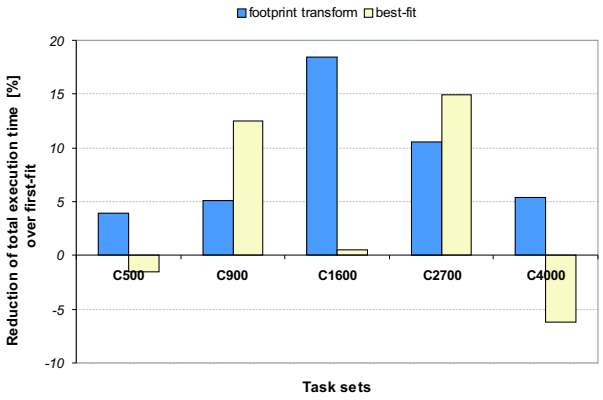

Figure 10. Reduction in execution time for best-fit and footprint transform over first-fit.

Figure 10 displays the reduction in execution time for the overall task sets relative to first-fit. Best-fit performs sometimes better than first-fit, sometimes worse. Applying footprint transforms leads to improvements in the overall execution time for all task sets. If the task sizes are rather small compared to the size of the array (C_{500}), the improvement is small because such tasks easily find first-fits. When the tasks get rather large (C_{4000}), the improvement decreases because for such tasks even footprint transform cannot find feasible placements. Besides the placement quality, the runtime of the placement technique is very important. First-fit is the computationally simplest technique, best-fit and footprint transform are clearly more complex. From our results it can be concluded that time spent for a complex placement technique is best invested in footprint transform, as this technique gives sound and substantial improvements.

5 Summary and Future Work

In this paper, we have made the case for the use of 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 node that receives tasks via the wireless interface,
- a first WURM-OS that includes task and resource management for hardware and software tasks, and
- a body area computing system consisting of a main module and a number of WURM nodes.

References

[1] C. Herring. Microprocessors, microcontrollers, and systems in the new millennium. *IEEE Micro*, 20(6):45–51, November/December 2000.

[2] S. S. Yau and F. Karim. Reconfigurable context-sensitive middleware for ADS applications in mobile ad hoc network environments. In *Proc. Int. Symp. on Autonomous Decentralized Systems (ISADS'01)*, pages 319–326, 2001.

[3] O. Mencer, M. Morf, and M. J. Flynn. Hardware software tri-design of encryption for mobile communication units. In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'98)*, volume 5, pages 3045–3048, 1998.

[4] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. Rabaey. Evaluation of a low-power reconfigurable DSP architecture. In *Proc. Reconfigurable Architectures Workshop (RAW'98)*, volume 1388 of *Lecture Notes in Computer Science*, pages 55–60. Springer, 1998.

[5] G. Stitt, B. Grattan, J. Villarreal, and F. Vahid. Using on-chip configurable logic to reduce embedded system software energy. In *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'02)*, 2002.

[6] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr., D. Patel, and S. Roundy. PicoRadio supports ad hoc ultra-low power wireless networking. *IEEE Computer*, 33(7):42–48, July 2000.

[7] Zhu Liu, Yao Wang, and Tsuhan Chen. Audio feature extraction and analysis for scene segmentation and classification. *The Journal of VLSI Signal Processing*, 20(1/2):61–79, October 1998.

[8] K. Van Laerhoven, K. A. Aidoo, and S. Lowette. Real-time analysis of data from many sensors with neural networks. In *Proc. Int. Symp. on Wearable Computers (ISWC'01)*, pages 115–122, 2001.

[9] Xilinx, Inc. PicoBlaze 8-bit microcontroller for Virtex devices. Application Note XAPP213, April 2002.

[10] Xilinx, Inc. *MicroBlaze Hardware Reference Guide*, March 2002.

[11] J. Beutel and O. Kasten. A minimal Bluetooth-based computing and communication platform. Technical report, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, 2001.

[12] R. Enzler, M. Platzner, C. Plessl, L. Thiele, and G. Tröster. Reconfigurable processors for handhelds and wearables: Application analysis. In *Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications III*, volume 4525 of *Proceedings of SPIE*, pages 135–146, 2001.

[13] M. Dyer and M. Wirz. Reconfigurable system on FPGA. Master's thesis, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, March 2002.

[14] J. Gaisler. *The LEON Processor User's Manual, Version 2.3.7*. Gaisler Research, August 2001.

[15] M. Lerjen and C. Zbinden. Reconfigurable Bluetooth–Ethernet bridge. Master's thesis, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, 2002.

[16] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design & Test of Computers*, 17(1):68–83, January–March 2000.

[17] H. Walder and M. Platzner. Non-preemptive multitasking on FPGA: Task placement and footprint transform. In *Proc. Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA'02)*, pages 24–30, June 2002.