Implementation of a HiperLAN/2 Receiver on the Reconfigurable Montium Architecture

Paul M. Heysters, Gerard K. Rauwerda, Gerard J.M. Smit
University of Twente, Department EEMCS,
PO Box 217, 7500 AE, Enschede, The Netherlands
{heysters, rauwerda, smit}@cs.utwente.nl

Abstract

A heterogeneous System-on-Chip (SoC) architecture for mobile hand-held devices is proposed to overcome the battery bottleneck in these devices. This SoC contains processing tiles of different granularities. The Montium coarse-grain reconfigurable tile processor is presented. Also, an introduction to HiperLAN/2 baseband processing is given. The implementation of a HiperLAN/2 receiver on the Montium reconfigurable architecture is explained in detail. The hardware of this implemented receiver has been simulated and the performance figures are given. The configuration overhead for the receiver is very small, which enables dynamic reconfiguration. The required computational performance can be obtained at very low clock frequencies. The Montium coarse-grain reconfigurable architecture enables an energy and area efficient implementation of a HiperLAN/2 receiver.

1 Introduction

The limited battery capacity in mobile hand-held devices has long since become one of the biggest system resource constraints. Applications of mobile devices (i.e. wireless access and multi-media processing) demand ever more computational power. Therefore, it is important to use energy-efficient hardware in mobile devices. Dedicated hardware (i.e. an ASIC) is the most energy-efficient. The drawbacks of dedicated hardware are the costs and the inflexibility. The development costs of an ASIC make it only feasible for very large volumes. Mobile devices need a high degree of flexibility. For example, in different regions, different wireless standards are used, or in the same region multiple wireless access technologies with different costs and quality of service might coexist. Standards (both for wireless access and multi-media) also evolve over time and user applications change even more frequently. A mobile device therefore requires the flexibility of a general-purpose processor (GPP). However, the power requirements of a GPP with enough performance to process the wireless access and multi-media applications render it useless for mobile hand-held devices.

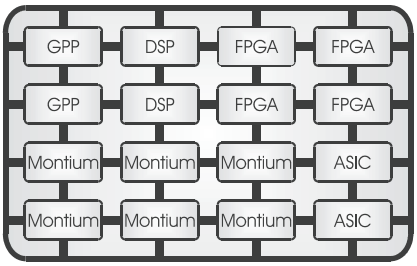

Figure 1: *The Chameleon SoC template.*

To overcome the contradicting performance, flexibility and energy requirements of mobile hand-held devices, we propose the 'Chameleon' system-on-chip (SoC) template. This SoC contains small heterogeneous processing tiles that are connected by an on-chip network. Each processing tile is specialized for a particular algorithm domain. The general philosophy is that an algorithm should be matched with hardware that can perform it efficiently. In the template four processor types are distinguished: *general-purpose* (i.e. general-purpose processors and digital signal processors), *fine-grain reconfigurable* (i.e. FPGAs), *coarse-grain reconfigurable* (e.g. Montium tiles) and *application specific* (i.e. ASICs). An example of a Chameleon SoC is depicted in Figure 1.

Coarse-grain reconfigurable architectures might be the key technology to obtain a good balance between flexibility and energy-efficiency. We have designed a coarse-grain reconfigurable processing tile called Montium. The target algorithm domain of the Montium tile processor (TP) comprises 16-bit digital signal processing algorithms that contain multiply accumulate operations such as FFT, FIR and linear interpolation.

This paper shows the implementation of the base band part of a HiperLAN/2 receiver on reconfigurable Montium TPs. HiperLAN/2 is a wireless local area network access technology in the 5 GHz frequency band. HiperLAN/2 uses orthogonal frequency division multiplexing (OFDM) modulation with a time division multiple access (TDMA) scheme.

2 Related work

Both academy and industry show interest in coarse-grained reconfigurable architectures. The Pleiades project at UC Berkeley [1] focuses on an architectural template for ultra low-power high-performance multimedia computing. In the Pleiades architecture template a general-purpose microprocessor is surrounded by a heterogeneous array of autonomous, special-purpose satellite processors. The Pleiades SoC design methodology assumes a (very) specific algorithm domain. A processor for speech coding applications, called Maia, was designed using the Pleiades architecture template. Quicksilver's adaptive computing machine (ACM) [6] technology is intended for low-power mobile devices. Their key observation is that algorithms are heterogeneous by nature. The architecture of the ACM comprises heterogeneous nodes of different granularities. The extreme processor platform (XPP) of PACT [3] is based on clusters of coarse-grained processing array elements (PAEs), which is either an ALU or a memory. Actual PAEs are tailored to the algorithm domain of a particular XPP processor. Silicon Hive [11] offers coarse-grain reconfigurable block accelerators (e.g. Avispa and Moustique) and stream accelerators (e.g. Bresca) for high performance and low power applications. The architecture consists of VLIW-like datapath elements called process storage elements (PSEs).

Much work has been done on Software Defined Radio in the SDR forum context [10]. However this forum mainly focuses on general-purpose processors and they do not concentrate on reconfigurable platforms and energy-efficiency. The Software Defined Radio project [12] at the University of Twente in the Netherlands aims to develop a receiver front-end capable of receiving all WLAN standards. In the Information Society Technologies (IST) EASY project [4] the objective is to develop a cost/power efficient heterogeneous SoC processor, which handles the baseband processing of 5 GHz WLAN systems. The heterogeneous SoC consists of embedded FPGA and an ARM processor.

3 Montium reconfigurable architecture

The Montium [7] – a descendent of the Field Programmable Function Array – targets the 16-bit digital signal processing (DSP) algorithm domain. A single Montium processing tile is depicted in Figure 2. At first glance the Montium architecture bears a resemblance to a VLIW processor. However, the control structure of the Montium is very different. For (energy-) efficiency it is imperative to minimize the control overhead. This can be accomplished by statically scheduling instructions as much as possible at compile time.

The lower part of Figure 2 shows the Communication and Configuration Unit (CCU) and the upper part shows the reconfigurable Tile Processor (TP). The CCU implements the interface for off-tile communication. The definition of the off-tile interface depends on the interconnect technology that is used in the SoC.

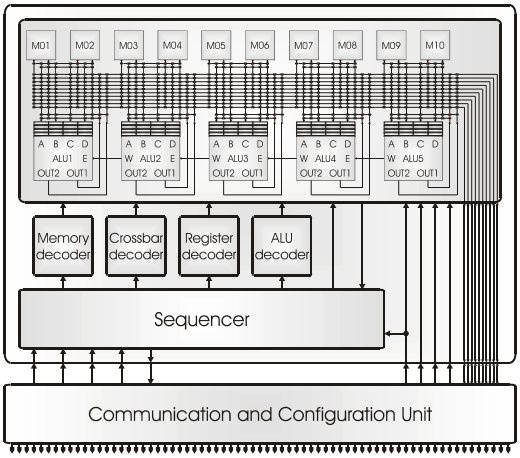

Figure 2*: The Montium processing tile.*

The TP is the computing part that can be configured to implement a particular algorithm. Figure 2 reveals that the hardware organization of the tile processor is very regular. The five identical ALUs (ALU1…ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01…M10) in parallel. The small local memories are also motivated by the locality of reference principle. The datapath has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. The ALU input registers provide an even more local level of storage. Locality of reference is one of the guiding principles applied to obtain energy-efficiency in the Montium. A vertical segment that contains one ALU together with its associated input register files, a part of the interconnect and two local memories is called a Processing Part (PP). The five Processing Parts together are called the Processing Part Array (PPA). A relatively simple sequencer controls the entire PPA. The sequencer selects configurable PPA instructions that are stored in the decoders of Figure 2.

Each local SRAM is 16-bit wide and has a depth of 512 positions, which adds up to a storage capacity of 8Kbit per local memory. A reconfigurable Address Generation Unit (AGU) accompanies each memory. It is also possible to use the memory as a lookup table for complicated functions that cannot be calculated using an ALU, such as sine or division (with one constant). A memory can be used for both integer and fixed-point lookups.

A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. An ALU has two 16-bit outputs, which are connected to the interconnect. The ALU is entirely combinational and consequentially there are no pipeline registers within the ALU. Neighbouring ALUs can also communicate directly on level 2. The West-output of an ALU connects to the East-input of the ALU neighbouring on the left. The East-West connection does not introduce a delay or pipeline, as it is not registered.

4 HiperLAN/2

HiperLAN/2 [5] is a wireless local area network (WLAN) access technology and is similar to the IEEE 802.11a WLAN standard. HiperLAN/2 operates in the 5 GHz frequency band and makes use of a method called orthogonal frequency division multiplexing (OFDM) to transmit the analogue signals. OFDM is efficient in office environments, where the transmitted radio signals are reflected from many points, leading to different propagation times before they reach the receiver. The air interface is based on time division multiple access (TDMA) and time division duplex (TDD). In a TDMA system a medium access control (MAC) frame is divided in multiple time slots. The bit rate of HiperLAN/2 at the physical level depends on the modulation type and is either 12, 24, 48 or 72 Mbit/s. Redundancy introduced by forward error correction results in an actual transmission rate of either 6, 9, 12, 18, 27, 36 or 54 Mbit/s.

The task of the physical layer in HiperLAN/2 is to modulate bits that originate from the data link control layer on the transmitter side and to demodulate them on the receiver side. The transmission format on the physical level is a burst, which consists of a preamble part and a data part. The basic idea of OFDM is to transmit high data rate information by dividing the data into several parallel bit streams, and let each one of these bit streams modulate a separate subcarrier. A HiperLAN/2 channel contains 52 subcarriers and has a channel spacing of 20 MHz. 48 subcarriers carry actual data and 4 carry pilots, which facilitate phase tracking for coherent demodulation.

In the upper part of Figure 3 a model of a HiperLAN/2 transmitter is shown. In the mapping part of the transmitter, groups of bits are mapped to complex-number values. Several modulation alternatives are provided to accomplish this: binary phase shift keying (BPSK), quadrature phase shift keying (QPSK) and quadrature amplitude modulation (QAM). In the OFDM part of the transmitter 48 of these complex-number values are, together with 4 pilot values, converted to 64 complex-number time samples. Finally, the transmitter inserts a prefix, which is an exact copy of the last 16 complex-number samples. So, one OFDM symbol consists of $16 + 64 = 80$ complex-number samples. The sample rate is 20 MHz and hence the duration of one OFDM symbol is 4 µs. For every MAC frame, the physical layer transmits a preamble – a sequence of known OFDM symbols – before the OFDM symbols containing the actual data are transmitted.

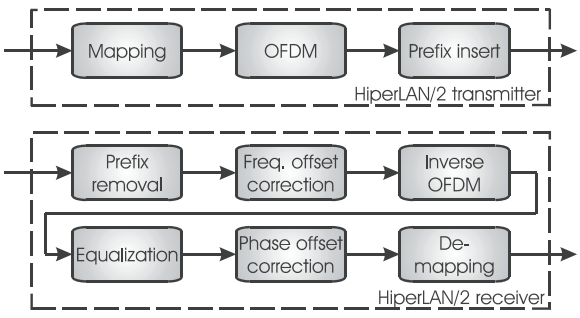

Figure 3: *HiperLAN/2 transmitter and receiver.*

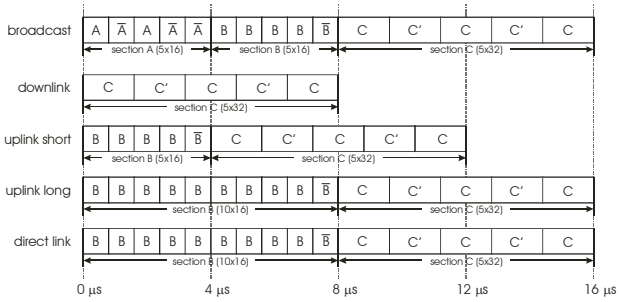

Figure 4: *Preambles for all HiperLAN/2 burst types.*

A HiperLAN/2 receiver performs the inverse of the transmitter in order to recover the transmitted bitstream. The receiver also has to correct any distortions caused by the radio channel and the hardware of the transmitter and receiver. A model of a HiperLAN/2 receiver is depicted in the bottom part of Figure 3.

A MAC frame always starts with a preamble. A preamble is a sequence of predefined OFDM symbols. The receiver uses the preamble to estimate the various distortions in the received signal. Which preamble is used depends on the burst type. In the HiperLAN/2 standard five burst types are defined: broadcast, downlink, uplink with short preamble, uplink with long preamble and direct link. All burst types use one or more preamble sections to precede the data burst. The following preamble sections are defined: A, B (short), B (long) and C. Figure 4 depicts how the preamble sections are used for each burst type.

The receiver first needs to synchronise with the transmitter. It does this by detecting the start of a transmission and then detecting the preamble sections by means of correlating the received stream of samples with the known preamble.

Because the sampling clock of the receiver hardware is not synchronised with the clock in the transmitter, the OFDM symbol window in the receiver may slowly wander away from the ideal OFDM symbol window. Therefore, the start position of the data of each received OFDM symbol containing data has to be determined. The prefix information of an OFDM symbol is used for this purpose. The maximum of a sequence of correlations between 16 samples and 16 samples received 64 samples earlier reveals the location of the prefix. The prefix is then removed from the OFDM symbol.

The difference in mix frequencies between the transmitter and receiver is called frequency offset and causes inter-subcarrier interference. It is assumed that the frequency offset does not change during the time of a MAC frame (i.e. 2 ms). The receiver can compensate for distortion caused by frequency offset by multiplying the 64 complex-number data samples of an OFDM symbol with the value of the frequency offset. The frequency offset can be determined by using information from the received preamble sections.

In the inverse OFDM part of the receiver the 64 complex-number time samples are converted to 48 received complex-number data values and 4 received (complex-number) pilot values. The received complex-number values may still suffer

from distortions that need to be corrected before de-mapping them to a bitstream.

The reflections of the transmitted radio signals have different propagation times before they reach the receiver. Slow movement of the receiver introduces a Doppler shift distortion in the received signal. These two effects together result in frequency selective fading. Also, since the transmitter and receiver are not synchronised, there will be a (constant) phase offset between them. An equalizer can compensate all these effects. The coefficients for the equalizer can be determined by using information from the received preamble sections. Since the coherence time of a HiperLAN/2 channel is about 20 ms and a burst of a MAC frame has a duration of 2 ms, the coefficients need to be determined only at the start of the frame [2].

The various distortions of the transmitted signal caused by a (realistic) wireless channel make it very difficult to estimate the frequency offset exactly at the receiver side. Therefore, the signal will still experience a small frequency offset. This small frequency offset causes a small, linearly changing difference in the phase of the transmitted and received signals. This phase offset is small and is assumed to be constant during one OFDM symbol. A phase offset corrector can detect this small phase offset by comparing the received pilot values with their predefined value and subsequently compensate for it.

The last stage of the receiver is the de-mapping of the complex-number values to a bitstream. The de-map function assumes that the most likely symbol to be transmitted was the symbol that maps (given the modulation type) to the complex-number value closest to the received complex-number value. This method of de-mapping is called hard decision de-mapping.

Table 1: *Parameters of typical channel models.*

Channel type	τ_{RMS} [ns]	Environment	Line of sight
A	50	office	no
B	100	open space / office	no
C	150	large open space	no
D	140	large open space	yes
E	250	large open space	no

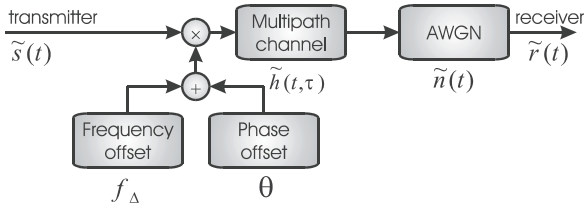

Figure 5: *Simulation model of a HiperLAN/2 channel.*

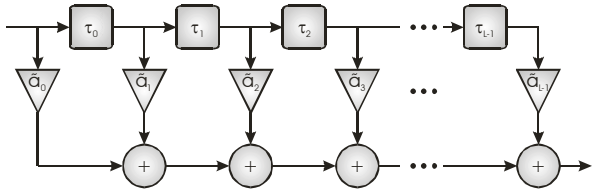

Figure 6: *Tapped delay line.*

5 Wireless channel

Figure 5 shows the wireless channel model used in our experiments. The wireless channel introduces multi-path and additive white Gaussian noise (AWGN). To speed up the simulation time, the simulation sampling rate is equal to the HiperLAN/2 baseband sampling rate of 20 MHz instead of the transmission rate of 5 GHz. Since in our simulation environment the transmitter and receiver are synchronised, the wireless channel models frequency offset and phase offset explicitly. Note that the sampling clock of the transmitter and receiver are still synchronised. The latter means that we assume that there is no symbol drift at the receiver side.

In [2] five types of channel models for HiperLAN/2 are given, which are derived from measurements in typical indoor and outdoor environments. These channel models have been defined for standardised system analysis [9]. The parameters of the typical channel models are given in Table 1 (in which τ_{RMS} is the RMS delay spread that characterises the relationship between the delay of a signal's path and the attenuation of that signal).

These channels can be implemented by means of a tapped-delay line model. The tapped-delay line model, as shown in Figure 6, simulates the behaviour of multi-path in a wireless environment. Each tap represents a path of the wireless signal travelling from transmitter to receiver. To each path a delay, τ_l, and a complex-number attenuation coefficient, $\tilde{a}_l$, is associated, corresponding to the delay and attenuation of the realistic signal path. The taps are assumed to be statistically independent and complex Gaussian distributed with zero mean. Except for the first tap, which can have a Ricean distribution for channel D, all taps have Rayleigh fading statistics.

In each path the signal also experiences Doppler shift due to motion in the environment. Such motion leads to frequency shifts in the signal's spectrum of individual reflected signals. These frequency shifts can be seen as time varying phase shifts. At the antenna of the mobile receiver many reflected signals arrive, all with different phase shifts. Thus, their relative phases change all the time and the amplitude of the resulting composite signal is affected. So, the Doppler effects determine the rate at which the amplitude of the resulting composite signal changes. Coherence time is a statistical measure of the time duration over which the channel impulse response is invariant. We assume that the maximum speed of the terminal is 3 m/s, which results in a maximum Doppler shift of 52 Hz. The Doppler spread and coherence time are inversely proportional to each other, yielding a coherence time of 20 ms. Hence, the channel can

be assumed constant during the time of one MAC frame of 2 ms.

Introducing all the channel effects, as depicted in Figure 5, will result in the received baseband signal before sampling:

$$\tilde{r}(t) = \left[\tilde{s}(t)e^{-j2\pi f_\Delta t+\theta} * \tilde{h}(t,\tau)\right] + \tilde{n}(t), \quad (1)$$

with $\tilde{s}(t)$ the originally transmitted baseband signal. f_Δ denotes the frequency offset, θ the common phase offset and the introduced white Gaussian noise is given by $\tilde{n}(t)$. * means that a convolution is performed. The time-variant channel impulse response, $\tilde{h}(t,\tau)$, denotes the multi-path contribution:

$$\tilde{h}(t,\tau) = \sum_{l=0}^{L-1} \tilde{a}_l(t)\delta(\tau - \tau_l), \quad (2)$$

where τ_l are the delays, $\tilde{a}_l$ are the complex-number amplitudes of the taps, δ is the Dirac pulse and L is the number of taps in the tapped-delay line. The channel realisations in Table 1 are modelled with $L = 18$ taps.

6 Implementation

The physical layer of a HiperLAN/2 receiver has been implemented on three Montium TPs. Figure 7 shows the functions that have been implemented on each Montium TP. Although prefix removal can be implemented in a Montium TP, it has not been implemented yet. In our simulation environment the sampling clock of the transmitter and receiver are synchronised. Therefore, there is no symbol drift at the receiver side and determining the location of the prefix of the OFDM symbol becomes trivial.

The receiver has been tested by means of a hardware simulator. Performance results of the Montium implementation of the receiver can be found in Section 7. Some of the receiver's functionality is performed in software (i.e. on a GPP). Irregular and computationally mellow tasks, which are outside the algorithm domain of the Montium, are performed in software. Note that this causes some communication overhead between the GPP and the Montium TPs. The irregular processes in the HiperLAN/2 receiver are the computing of coefficients for frequency offset correction and equalization. These coefficients have to be determined only once per MAC frame, i.e. once per 2 ms. Table 2 shows the computational complexity of the implemented receiver functions in terms of multiply and addition operations per MAC frame. The block size of the data the functions operate on is expressed in pairs of 16-bit words that represent a complex-number value. The table clearly demonstrates that the bulk of the multiplications and additions are performed on the reconfigurable hardware. Note that no multiplications or additions are required to determine the equalizer coefficients; they are determined by means of 52 complex-number divisions.

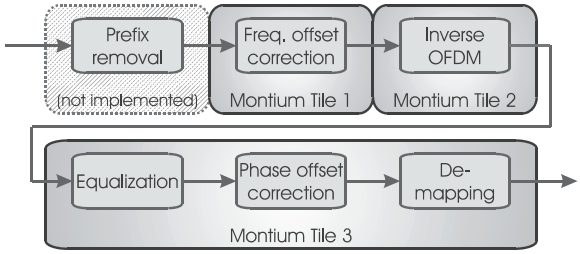

Figure 7: *Partitioning of a HiperLAN/2 receiver on three Montium Tile Processors.*

Table 2: *Reconfigurable hardware/software partitioning.*

Function	Implemented in	Block size	Multiplies per MAC frame	Additions per MAC frame
Determine frequency offset	Software	32	64	64
Determine equalizer coefficients	Software	52	0	0
Prefix removal	-	80	-	-
Frequency offset correction	Montium	64	127,744	95,309
Inverse OFDM	Montium	64	383,232	574,848
Equalizer, Phase offset, de-map	Montium	52	203,184	104,082

6.1 Frequency offset corrector

The frequency offset corrector is implemented in one Montium TP. One complete OFDM symbol can be corrected for frequency offset in 67 clock cycles. In one OFDM symbol 64 complex-number time samples are multiplied with a complex-number coefficient. This complex-number coefficient is determined once per MAC frame in software (i.e. on the GPP).

The frequency offset correction coefficient is estimated by computing the cross-correlation of preamble C [2].
Figure 4 shows that preamble C consists of a prefix of 32 samples and 2 equal sequences of 64 samples. The sequence consists of 2 parts (C and C'), which are mirrored. The first 16 samples of the two sequences are cross-correlated. The angle of the result of the complex-number correlation determines the total phase offset over 64 samples. The phase offset ϕ is linearly proportional with the frequency offset f_Δ (see equation (1)):

$$\phi = 2\pi f_\Delta t \Big|_{\phi\in[-\pi,\pi)} \Leftrightarrow f_\Delta = \frac{\phi}{2\pi t}, \quad (3)$$

The cross-correlation computes the total phase offset for 64 samples, which corresponds to an interval of $64 \times 50 \cdot 10^{-9} = 3.2$ µs. The maximum frequency offset, f_Δ, that can be detected in this way is ±156 kHz.

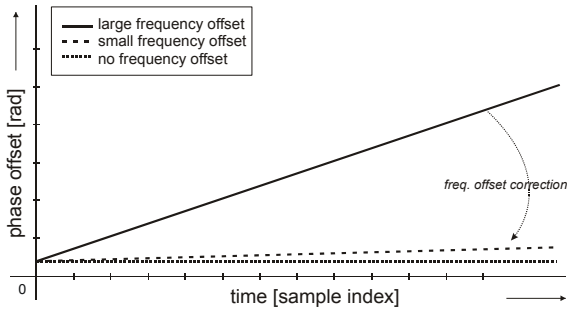

Figure 8: *The effect of frequency offset correction on phase offset.*

The phase offset between two samples is stored in the Montium's memory. Based on this phase offset and the index of the data sample, the Montium determines the complex-number offset correction coefficient with its look-up table (LUT) functionality. After the LUT operation the complex-number offset correction coefficient is multiplied with the corresponding complex-number data sample.

The frequency offset causes the phase of the received baseband signal to increase (or decrease) linearly with time, which can be observed from equations (1) and (3). Ideal frequency offset correction would result in a phase of the baseband signal that is constant over time. However, in real-life the frequency offset cannot be determined exactly. As a result, the phase of the signal after correction is still a little time-variant. This is illustrated in Figure 8. The time-variant phase needs to be corrected by a phase offset corrector. We assume that the slightly changing phase offset is almost constant during one OFDM symbol of 4 μs. Using the pilot symbols the phase offset corrector has to be trained every OFDM symbol.

6.2 Inverse OFDM

A Fast Fourier Transform (FFT) on a vector of 64 complex-number time samples can perform the inverse OFDM function. The FFT algorithm was already available for the Montium TP.

6.3 Equalizer, phase offset corrector and de-mapper

The equalizer coefficients are determined at the start of each MAC frame. The channel impulse response is assumed to be time-invariant during one complete MAC frame, because the coherence time of the wireless channel (20 ms) is much longer than the time of a MAC frame (2 ms). Therefore, the equalizer is trained only once per MAC frame.

The equalizer coefficients are determined by dividing the predefined preamble C values by the received complex-number values. Note that the received complex-number preamble values are obtained by applying frequency correction and inverse OFDM to the received time samples. Both operations are done by the Montium as explained in the previous sections. The computation of the equalizer coefficients themselves is done by a GPP. This function is implemented in software because it occurs infrequently (once every 2 ms) and because the Montium is not optimised for complex-number division operations. Once the GPP has computed the complex-number equalizer coefficients, they are stored in the memory space of the Montium TP that implements the equalizer.

During equalization each of the 52 complex-number subcarrier values is multiplied with its corresponding complex-number coefficient. The Montium TP can compute a complex-number multiplication in a single clock cycle using four (of the five) ALUs. The 48 equalized complex-number data values are temporarily stored in the local memories. These values still need to be corrected for phase offset. Before the phase offset can be corrected, the phase offset correction coefficient needs to be determined.

The four received equalized pilot values are not stored in memory; they are used to determine the phase offset coefficient. Because the phase offset coefficient needs to be determined for every OFDM symbol, it is implemented in hardware (i.e. on the Montium). Computing the phase offset coefficient in software is very expensive due to the frequent communication between GPP and Montium tile.

The phase offset coefficient $\tilde{C}_{po}$ is the mean correction factor for the received pilot values relative to the expected pilot value:

$$\tilde{C}_{po}(i) = \frac{P_d(i)}{\frac{1}{4}\left(\tilde{P}_{r1}(i) + \tilde{P}_{r2}(i) + \tilde{P}_{r3}(i) - \tilde{P}_{r4}(i)\right)}, \qquad (4)$$

where $\tilde{P}_{r1}(i)$, $\tilde{P}_{r2}(i)$, $\tilde{P}_{r3}(i)$ and $\tilde{P}_{r4}(i)$ are the received complex-number pilot values in OFDM symbol i. Note that the sign of $\tilde{P}_{r4}(i)$ is inverted. $P_d(i)$ is the predefined (real-number) pilot value for OFDM symbol i. This list is stored in memory of the Montium TP. The functionality of equation (4) and the equlization of the pilot symbols is implemented on the Montium in 8 clock cycles.

The phase offset correction itself is a complex-number multiplication of each equalized data value – which is already stored in a local memory of the Montium TP – with the phase offset coefficient $\tilde{C}_{po}$.

After the phase offset correction, a received complex-number value is de-mapped. Phase offset correction and de-mapping are performed concurrently in a pipelined fashion. The de-mapping itself is implemented by means of a single look-up table. For each HiperLAN/2 modulation alternative (i.e. BPSK, QPSK, 16-QAM and 64-QAM) a specific de-map table can be defined. The look-up table stores the combined look-up information for both the real and the imaginary received value. So, the index in this table contains a part for both the real and the imaginary value. There are four parameters available to fine-tune the computation of the index for the de-map table. The de-map table and parameters all reside in the memories of the Montium tile. By replacing the look-up table and modifying the parameters, the modulation scheme can be changed without reconfiguring the Montium TP

6.4 Discussion

The Montium TP that implements the FFT for the inverse OFDM function has the worst execution time (204 clock cycles) of the three tiles, as can be seen in Table 3. The FFT has to be performed for every OFDM symbol, that is once every 4 μs. For a fair impression of the required clock frequency the communication overhead between tiles should also be taken into account. In Table 3 various inter-tile communication schemes are considered. The assumed network-on-chip capacity is specified between brackets. To meet the HiperLAN/2 real-time requirements the FFT tile has to run at least at 80 MHz in worst case or at least at 51 MHz in case of a streaming FFT implementation. A Montium realized in a 0.12 μm CMOS technology can achieve this performance for the FFT. The maximum clock frequency of a Montium depends on the critical path through the ALUs and ranges from 45 to 150 MHz.

The area of a single Montium TP (excluding CCU) is about 2 mm^2 in 0.12 μm CMOS technology. To save area the frequency offset correction and the inverse OFDM function can be combined in the same tile. However, the execution time for this combined functionality would increase to about 67+204=271 clock cycles. The required clock frequency would be at least 97 MHz (worst case).

The flexibility of a reconfigurable architecture comes at the price of the configuration overhead. A Montium TP has to be configured with an algorithm before it can do useful work. This is a one-time overhead for the lifetime of the algorithm. Typically, the configuration time is amortized over the execution time of the algorithm. The configuration overhead for a coarse-grain architecture like the Montium is relatively small. Table 3 also shows the configuration size for each of the implemented functions. The Montium is configured via a 16-bit wide configuration interface. Because of the small configuration size, the configuration time is also very short. Consider that a Montium TP is configured with the frequency offset correction function. At 100 MHz, the configuration time is only 1.37 μs; short enough to classify as dynamic reconfiguration.

Table 3: *Properties of the Montium implementations.*

	Frequency offset correction	Inverse OFDM	Equalizer, Phase offset, de-map
Execution time in clock cycles	67	204	110
Communication time in clock cycles (32-bits/clock)	128	116	100
Minimal clock frequency in MHz (32-bits/clock)	49	80	53
Minimal clock frequency in MHz (32-bits/clock@100MHz)	25	72	37
Minimal clock frequency in MHz (64-bits/clock streaming)	17	51	28
Configuration size in bytes	274	946	576
Configuration time in clock cycles	137	473	288

7 Simulation results

A reference model of this HiperLAN/2 receiver was implemented in Matlab. The Matlab reference model uses 64-bit floating-point computations, whereas the Montium uses 16-bit fixed-point computations. Simulations were performed with both the Montium implementation of the receiver and the reference model.

In each simulation a downlink burst of a complete MAC frame of data is received. One MAC frame contains 500 OFDM symbols. In a downlink burst, the first two OFDM symbols of a MAC frame contain preamble C. Preamble C is used to estimate the frequency offset correction and the equalization coefficients. The remaining 498 OFDM symbols contain the data. The data is mapped using the 16-QAM modulation scheme. In this way 498×48×4=95,616 bits are received per simulation. Hence, bit error rates (BERs) of up to 10^{-3} represent statistically valid simulation results.

The HiperLAN/2 receiver was simulated using an 'A' channel and using an 'E' channel (shown in Table 1). White Gaussian noise is added to the received signal. The receiver was also simulated using an AWGN channel without multi-path. The amount of noise added to the signal is expressed by the signal-to-noise ratio (SNR). The SNR is expressed in E_s/N_0. N_0 denotes the noise spectral density function and E_s describes the signal energy per symbol. Furthermore, simulations were performed with varying frequency and/or phase offsets.

Figure 9 shows BER curves for both the reference model and the Montium implementation. Results for an 'A', an 'E' and an ideal channel with AWGN are shown. The performance of the floating-point reference model and the Montium implementation are about the same. The difference in performance between the reference model and the Montium implementation is less than 0.5% and falls within the simulation error.

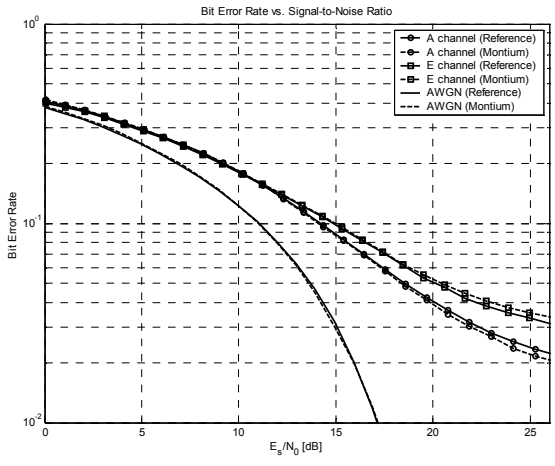

Figure 9: *Performance of the HiperLAN/2 receiver for different wireless channels.*

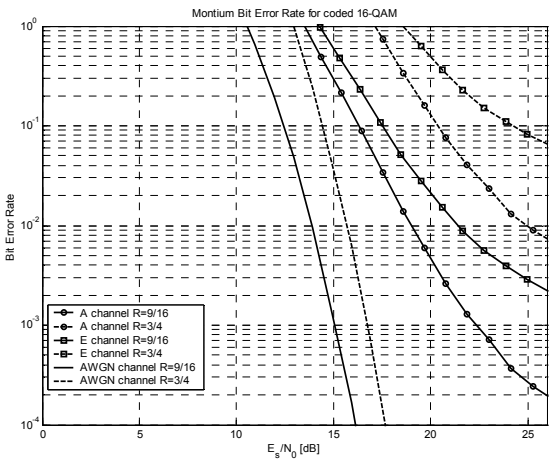

Figure 10: *Bit error rates for 16-QAM modulation after error correction for different channel models.*

Figure 9 shows the raw BER before error correction, since only the physical layer of the receiver was implemented. The upper bounds for the BER after error correction are given in Figure 10. The curves after error correction are estimated according to [13]. The HiperLAN/2 receiver needs a BER of $2.4*10^{-3}$ in order to reach the minimum defined sensitivity of 10% packet error rate [5]. The upper bounds show that the minimum sensitivity can be met with the Montium implementation.

The BER performance of the Montium implementation compares well with the simulation results in [8]. In order to realize the minimum required 10% packet error rate the Montium implementation requires a channel with a SNR (expressed in E_s/N_0) that is only 1 dB better than the simulation results in [8].

8 Conclusion

The motivation for our work is the increasing gap between the required performance and the available energy in mobile hand-held devices. As a solution, we proposed a heterogeneous SoC, which consists of domain specific tiles. The Montium is a coarse-grain reconfigurable tile processor for digital signal processing algorithms. In this paper the implementation of a HiperLAN/2 receiver on the Montium architecture and its simulation results were discussed.

A HiperLAN/2 receiver can be implemented in three Montium tile processors. These tile processors can meet the real-time performance requirements at fairly low clock frequencies; typically ranging from 25 to 72 MHz, depending on the function. Also, the (one-time) configuration overhead is low; ranging from 274 to 946 bytes, depending on the function.

A reference model of the HiperLAN/2 receiver was implemented in Matlab. The Montium receiver was compared against this reference model. A standardised wireless channel model was implemented to enable a fair comparison with other HiperLAN/2 receivers.

The Montium HiperLAN/2 receiver implementation can realize the minimum required BER of $2.4*10^{-3}$ after error correction. Hence, a useful HiperLAN/2 receiver can be implemented on the Montium.

Acknowledgements

Many thanks to Roel Schiphorst and Fokke Hoeksema for sharing their HiperLAN/2 knowledge with us.

PROGRESS and Freeband Knowledge Impulse support this research. PROGRESS is the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW. Freeband Knowledge Impulse is a joint initiative of the Dutch Ministry of Economic Affairs, the Technology Foundation STW, knowledge institutions and industry.

References

[1] A. Abnous, “Low-Power Domain-Specific Processors for Digital Signal Processing”, *Ph.D Dissertation, University of California*, Berkeley, USA, 2001.

[2] A. Berno, “Time and frequency synchronization algorithms for HIPERLAN/2”, *MSc. Thesis, University of Padova*, Italy, October 2001.

[3] V. Baumgarte, F. May, A. Nückel, M.Vorbach & M. Weinhardt, “PACT XPP – A Self-Reconfigurable Data Processing Architecture”, *Proceedings Engineering of Reconfigurable Systems and Algorithms*, pp. 64-70, Las Vegas, USA, June 2001.

[4] EASY project, http://easy.intranet.gr.

[5] ETSI, “Broadband Radio Access Networks (BRAN); HiperLAN type 2; Physical (PHY) layer”, ETSI TS 101 475 V1.2.2 (2001-02), 2001.

[6] G. Heidari & K. Lane, “Introducing a Paradigm Shift in the Design and Implementation of Wireless Devices”, *Proceedings Wireless Personal Multimedia Communications*, vol.1 pp. 225-230, Aalborg, Denmark, September 2001.

[7] P.M. Heysters, G.J.M. Smit & E. Molenkamp, “A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems”, *The Journal of Supercomputing*, volume 26, number 3, Kluwer Academic Publishers, Boston, U.S.A., November 2003, ISSN 0920-8542.

[8] J. Khun-Jush, P. Schramm, U. Wachsmann & F. Wenger, “Structure and Performance of the HIPERLAN/2 Physical Layer”, *Proceedings VTC ’99 - Fall*, pp. 2667-2671, Amsterdam, the Netherlands, September 1999.

[9] J. Medbo, P. Schramm, “Channel Models for HIPERLAN/2”, ETSI/BRAN 3ERI085B, March 1998.

[10] SDR Forum, http://www.sdrforum.org.

[11] Silicon Hive, http://www.siliconhive.com.

[12] Software Defined Radio project, http://www.sas.el.utwente.nl/home/SDR.

[13] R.E. Ziemer & R.L. Peterson, “Introduction to digital communication”, second edition, Prentice Hall, USA, 2001.