Mapping DSP algorithms to a reconfigurable architecture

–

Adaptive Wireless Networking (AWGN)

Gerard Rauwerda
g.k.rauwerda@utwente.nl

University of Twente
faculty of Electrical Engineering, Mathematics & Computer Science
Computer Architecture, Design and Test for Embedded Systems group
P.O. Box 217, 7500 AE Enschede
the Netherlands

June, 2003

Abstract

This report will discuss the *Adaptive Wireless Networking* project. The vision of the *Adaptive Wireless Networking* project will be given. The strategy of the project will be the implementation of multiple communication systems in dynamically reconfigurable heterogeneous hardware. An overview of a wireless LAN communication system, namely HiperLAN/2, and a Bluetooth communication system will be given. Possible implementations of these systems in a dynamically reconfigurable architecture are discussed. Suggestions for future activities in the *Adaptive Wireless Networking* project are also given.

Contents

List of Figures

List of Tables

Acronyms

ACL	Asynchronous Connection-Less
ADC	Analog-to-Digital Converter
AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
AM	Amplitude Modulation
ASIC	Application specific integrated circuit
BPSK	Binary Phase Shift Keying
CCU	Communications and Configuration Unit
CDMA	Code Division Multiple Access
CPU	Central processing unit
CRC	Cyclic Redundancy Check
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FEC	Forward Error Control
FH-CDMA	Frequency-hopping-CDMA
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
FPFA	Field Programmable Function Array
FSK	Frequency Shift Keying
GFSK	Gaussian FSK
GHZ	Giga Hertz
GPP	General Purpose Processor
HEC	Header Error Control
ISM-band	Industrial-, Scientific-, Medical-band
LPF	Low Pass Filter
MAC	Multiply accumulate
Mbps	Mega Bit per Second
MHz	Mega Hertz
MSPS	Mega Samples per Second
NRZ	Non-Return to Zero
OFDM	Orthogonal frequency division multiplexing

PP	Processing Part
PPA	Processing Part Array
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RX	Receive
SCO	Synchronous Connection-Oriented
SDR	Software Defined Radio
SoC	System-on-Chip
TDD	Time-Division Duplex
TX	Transmit
UMTS	Universal Mobile Telecommunications System
VLIW	Very Long Instruction Word

Chapter 1

Introduction

Traditionally, computations are either implemented in hardware or in software running on processors. More recently, however, new alternatives are introduced which mix properties of the traditional hardware and software alternatives. The class of these new alternatives is denoted as *reconfigurable computing*. This class of architectures is important because it allows the computational capacity of the machine to be highly customized to the instantaneous needs of an application while also allowing the computational capacity to be reused in time at a variety of time scales.

Reconfigurable Computing is emerging as an important new organizational structure for implementing computations. It combines the post-fabrication programmability of processors with the spatial computational style most commonly employed in hardware designs. The result changes traditional 'hardware' and 'software' boundaries, providing an opportunity for greater computational capacity and density within a programmable media.

This report addresses the basic issues involved in the design process for facilitating reconfigurable computing in communication systems. We aim at implementation of communication systems in reconfigurable hardware. One of the tasks is to map multiple communication systems on a platform of heterogeneous reconfigurable architectures. In chapter 2 we will give the outline of the *Adaptive Wireless Networking (AWGN)* project. Basics of reconfigurable computing will be given as well as the proposed dynamically reconfigurable heterogeneous architecture. Once the specifications of reconfigurable heterogeneous architecture are known, we can study the feasibility of implementing communication systems on this architecture. In chapter 3 a wireless LAN communication system, namely the HiperLAN/2 standard, will be discussed. The most relevant properties of the standard are given and suggestions for implementations of the HiperLAN/2 receiver are made. In chapter 4 a less complex communication system is considered. Also for the Bluetooth receiver suggestions for implementations are made. Finally in

chapter 5 a conclusion about mapping DSP algorithms to a reconfigurable architecture will be given based on the cases of a simple and a complex communication systems. Furthermore some recommendations and future research will be given.

Chapter 2

Reconfigurable computing

2.1 Introduction

Evolution of technology (e.g. in DSPs and reconfigurable computing) will allow to move digitization closer and closer to RF in the radio access network. Software Defined Radio (SDR) techniques will go ahead with availability of low-cost enabling components. SDR has a very high potential to enable a major leap for faster provision of more flexible, advanced mobile communication services. Multi-standard network elements in the radio access network combined with advanced network management will allow dynamically assigned services, QoS support for user applications, use the spectrum and network resources more efficient and will push integration of services and networks in the global, business and domestic environment. SDR may change the scope and role of standardization allowing more freedom for implementation of new services. For industrial companies active in this field the main driving forces for using SDRs are: cost reduction (avoiding costly re-designs of ASICs), flexibility (using the same hardware for different traffic patterns and new (or revised) standards) and time to market. We expect that the combination of high-level design tools and reconfigurable hardware architectures will enable designers to develop highly flexible, efficient and adaptive base stations and applications for future UMTS terminals. Performance and power gains will be achieved by applying dynamically reconfigurable (heterogeneous) architectures. In this approach application-specific chip-design (ASIC design) is replaced by dynamic reconfiguration and reprogramming. The technological challenges to realize SDR for a UMTS base station are huge:

- Highly efficient system architectures have to be defined that are scalable, flexible and adaptable to applications demanding differences in processing capacity requirements (QoS).

- The communication between the various processing entities has to be

high-performance, flexible and low-power.

- The middleware and (distributed) operating systems have to support real-time requirements and (distributed) multi-tasking capability with minor overhead.
- The mapping of the algorithms to the architecture has to be done carefully, as this is closely related to the efficiency of the system.

2.1.1 Adaptivity

A key issue of systems that have to support future mobile networks is that they have to be adaptive. These systems have to adapt to changing environmental conditions (e.g. more or less users in a cell or varying noise figures due to reflections or user movements) as well as to changing user demands (QoS). Furthermore, these architectures have to be extremely efficient as these are used in battery-operated terminals and cost effective as they are used in consumer products as well as in base stations. Although energy-efficiency is a major issue in terminals as they draw their energy from small batteries, energy consumption is also an issue in base stations from a technical (costly cooling of chips and power supplies) as well as from an environmental point of view.

2.1.2 Reconfigurability

There are quite a number of good reasons for using reconfigurable architectures in future wireless terminals and base stations:

- Although reconfigurable systems are known to be less efficient compared to ASIC implementations they can have considerable benefits. For example: wireless systems work in a very dynamic environment, this means that depending of the distance of the receiver and transmitter or cell occupation more or less processing power is needed. When the system can adapt – at run-time – to the environment significant savings can be obtained.
- Standards evolve quickly; this means that future systems have to have the flexibility and adaptivity to adapt to slight changes in the standards. By using reconfigurable architectures instead of ASICs costly re-designs can be avoided.
- 3G systems based on the UMTS standard have a QoS based transmission scheme i.e. they are highly flexible and adaptable to new services.
- Downloadable reconfigurations (long-term reconfiguration) enable new or adapted services on existing terminals.

- Traditional (DSP) algorithms are rather static. The recent emergence of new applications that require sophisticated adaptive, dynamical algorithms based on signal and channel statistics to extract optimum performance has drawn renewed attention to run-time reconfigurability.

2.2 Reconfigurable heterogeneous architectures

In the *Adaptive Wireless Networking (AWGN)* project we aim at implementation of communication systems in reconfigurable embedded systems. One of the tasks in the project is to map (different) communication systems on a platform of heterogeneous reconfigurable architectures. The platform is heterogeneous in the sense that digital signal processing is performed in general purpose processors (GPPs), bit-level reconfigurable hardware (i.e. FPGAs) and word-level reconfigurable hardware (i.e. MONTIUM).

Basically one can distinguish between different processor types in a heterogeneous architecture, based on the granularity of the operations:

- **Fine grained** operations in the modules that perform functions like multiply and addition. The operations are performed on bit-level. The processor type is denoted as reconfigurable logic, like an FPGA.
- **Medium grained** operations are the functions of the modules. The functional tasks are allocated to dedicated modules. The operations are performed on word-level. The processor type is denoted as reconfigurable data-path.
- **Course grained** operations are those tasks that are not specific for a module and that can be performed by the CPU module, or even on a remote compute server. The processors are denoted as general-purpose programmable units, like an GPP or DSP unit.

The granularity of the reconfigurable logic is the size of the smallest functional unit that is addressed by the mapping tool. Lower granularity provides more flexibility in adapting the hardware to the computation structure. A heterogeneous system combines performance, flexibility and energy-efficiency. It should support high performance through parallelism, should match the computational model of the algorithm to granularity of the processing entity, should operate at minimum supply voltage and clock frequency and should provide flexibility only where needed and desirable and at the right granularity.

In the *Chameleon* project [1] a dynamically reconfigurable heterogeneous System-on-a-Chip (SoC) is being defined. The SoC contains a general purpose processor, a bit-level reconfigurable part and several word-level reconfigurable parts. The proposed heterogeneous architecture is shown in figure

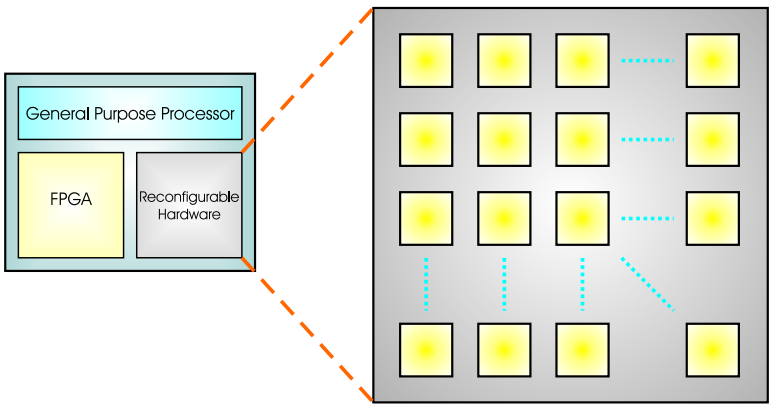

Figure 2.1: CHAMELEON heterogeneous SoC architecture

2.1. The programmability of the architecture enables the system to be targeted at multiple applications.

2.2.1 Fine granularity

Field programmable gate arrays (FPGAs) are useful for applications with bit-level operations. Many reconfigurable computing systems are based on FPGAs nowadays. Reconfigurable architectures have evolved from FPGAs. FPGAs consist of a matrix of logic blocks and interconnection network. The functionality of the logic blocks and the connections in the interconnection network can be modified by downloading bits of configuration data onto the hardware.

2.2.2 Medium granularity

The MONTIUM architecture, also known as field programmable function array (FPFA), is an example of a word-level reconfigurable data-path. The architecture consists of multiple interconnected processor tiles. The algorithm domain of the MONTIUM comprises 16-bit DSP algorithms that contain multiply accumulate (MAC) operations. A MONTIUM-tile is designed to execute highly regular computational intensive DSP kernels.

Figure 2.2 depicts a single MONTIUM processor tile. The hardware organisation within a tile is very regular and resembles a very long instruction word (VLIW) architecture. The five identical arithmetic and logic units (ALU1$\cdots$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$\cdots$M10) in parallel. The small local memories are also motivated by the locality of reference principle. 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 communication and configuration unit (CCU) implements the interface with the world outside the tile. The MONTIUM has a datapath width of 16-bits and supports both integer and fixed-point arithmetic. Each local SRAM is 16-bit wide and has a depth of 512 positions, which adds up to a storage capacity of 8 Kbit per local memory. A memory has only a single address port that is used for both reading and writing. A reconfigurable address generation unit (AGU) accompanies each memory. The AGU contains an address register that can be modified using base and modify registers.

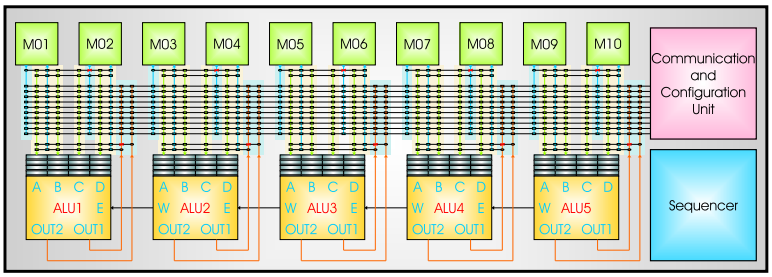

Figure 2.2: MONTIUM processor tile

It is also possible to use the memory as a lookup table for complicated functions that cannot be calculated using an ALU, such as sinus or division (with one constant). A memory can be used for both integer and fixed-point lookups. The interconnect provides flexible routing within a tile. The configuration of the interconnect can change every clock cycle. There are ten busses that are used for inter-PPA communication. Note that the span of these busses is only the PPA within a single tile. The CCU is also connected to the global busses. The CCU uses the global busses to access the local memories and to handle data in streaming algorithms. Communication within a PP uses the more energy-efficient local busses. 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 combinatorial and consequentially there are no pipeline registers within the ALU. The diagram of the MONTIUM ALU in figure 2.3 identifies two different levels in the ALU. Level 1 contains four function units. A function unit

implements the general arithmetic and logic operations that are available in languages like C (except multiplication and division). Level 2 contains the MAC unit and is optimized for algorithms such as FFT and FIR. Levels can be bypassed (in software) when they are not needed.

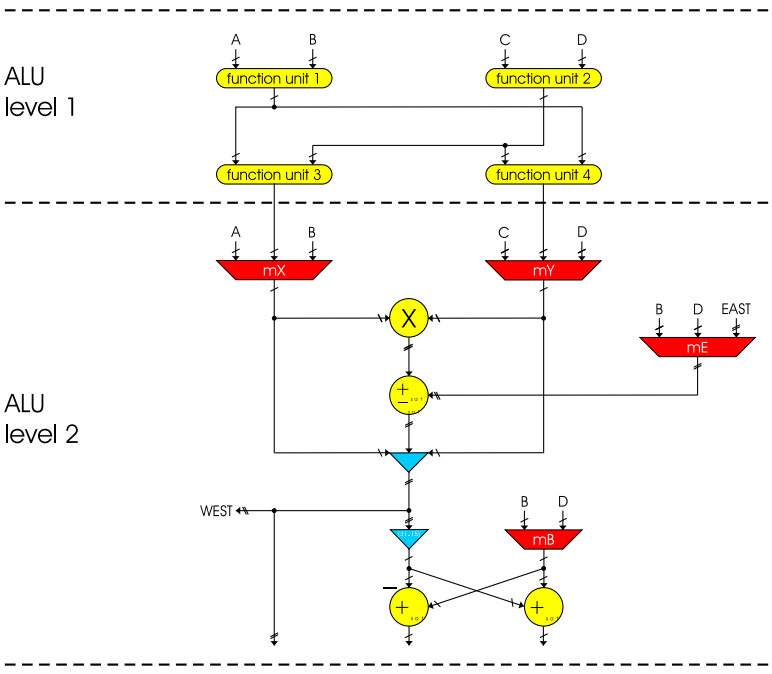

Figure 2.3: Montium ALU

Neighboring ALUs can also communicate directly on level 2. The West-output of an ALU connects to the East-input of the ALU neighboring on the left (the West-output of the leftmost ALU is not connected and the East-input of the rightmost ALU is always zero). The 32-bit wide East-West connection makes it possible to accumulate the MAC result of the right neighbor to the multiplier result (note that this is also a MAC operation). This is particularly useful when performing a complex multiplication, or when adding up a large amount of numbers (up to 20 in one clock cycle). The East-West connection does not introduce a delay or pipeline, as it is not registered.

2.2.3 Course granularity

General purpose processors (GPPs) can be programmed to perform all kinds of computational tasks. However, they have to pay for this flexibility with a high energy consumption. The energy overhead in making the architecture programmable most often dominates the energy dissipation of the intended

computation. However, general purpose processors are very good in control type of applications; e.g. applications with frequent control constructs (if-then-else or while loops).

Chapter 3

HiperLAN/2 physical layer

3.1 Introduction

In this chapter the implementation of a software defined HiperLAN/2 physical layer model will be described shortly, based on [2]. The model should provide insight in the demodulation functions that are necessary in HiperLAN/2 and it should be useful for determining channel selection and computational requirements for the software defined radio project at the University of Twente [3]. The model is implemented in Matlab Simulink and uses C++ as descriptive language.

Different important parts in the HiperLAN/2 receiver are considered and implemented in the model. We will try to map these important parts of the receiver on a MONTIUM architecture (also known as Field Programmable Function Array (FPFA)) [4]. Before the mapping can be performed, one has to know the specifications of all the signals, which have to be processed.

In section 3.2 some knowledge about the physical layer in HiperLAN/2 will be given. The most important receiver algorithms will be discussed in section 3.3. Once the receiver algorithms are known, one can investigate the computational requirements of these algorithms. In section 3.4 the computational requirements of these algorithms are studied considering the computation time, and the number of configurations of the data path. The consequences of mapping the receiver algorithms on the MONTIUM architecture are studied in section 3.5 of this paper. In section 3.6 we draw a little conclusion and do some recommendations.

3.2 Physical layer in HiperLAN/2

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 layer is a burst, which consists of a preamble and a data part.

The frequency spectrum available to HiperLAN/2 is divided into 19 so called channels, which are referred as radio channels. Each of those radio channels has a bandwidth of 20 MHz.

Orthogonal frequency division multiplexing (OFDM) has been chosen as modulation technique in HiperLAN/2. OFDM is a special kind of multi-carrier modulation. The modulation technique divides the high data rate information in several parallel bit streams and each of those bit streams modulates a separate subcarrier.

The physical layer transmits 52 subcarriers in parallel per radio channel. Four of the 52 subcarriers are used to transmit pilot tones. Those pilots assist the demodulation in the receiver.

3.3 Receiver model algorithms

The receiver not only has to convert the received signal to data bits by performing the inverse of the transmitter, but it also has to try to inverse distortions caused by the radio channel. The receiver can roughly be divided into two parts, a time domain part and a frequency domain part.

In the first stage of the receiver, signal functions will be time domain functions. In the second stage of the receiver, signal functions will be frequency domain functions. Most of the operations can be performed in time domain and in frequency domain. The location of the functions in the receiver architecture in [2] is based upon a trade-off between the necessary resolution that must be reached for a certain correction and the solution with the minimum number of operations. One also tried to keep the corrections independent of each other by deciding the execution order of the functions.

A HiperLAN/2 receiver should at least contain the following functions:

- synchronization function
- Frequency offset corrector
- Phase offset corrector
- Channel equalizer
- Inverse OFDM
- De-mapping
- De-interleaving *
- Viterbi-decoder *
- De-scrambling *

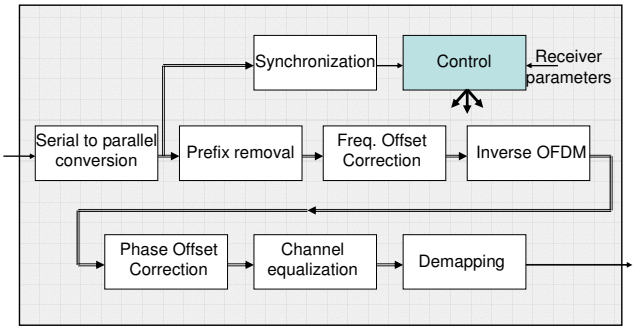

Figure 3.1: Demodulator part of the HiperLAN/2 receiver

In [2] one has implemented a HiperLAN/2 receiver with the functions as given in Figure 3.1. Those functions are being subjected to further research, in order to map the functions on a MONTIUM architecture. The functions in the list above that are denoted with an asterisk * are not implemented. The channel equalization function is partly implemented in the model, in that it is not adaptive.

3.3.1 Serial to parallel conversion

One OFDM symbol is represented by 80 complex input samples, so once per 80 samples the receiver will have enough information to demodulate the received samples and output the resulting bits. The sample rate of the input signal is 20 MHz, so the duration of an OFDM symbol is 4 μs. The main modulation part of the receiver, the inverse OFDM, works with parallel data instead of serial data. Hence a vector of at least 80 samples will be created, containing the oldest sample at the top of the vector and the newest sample at the bottom. A cyclic buffer can perform this function efficiently. We assume that each input sample represents a 16-bit fixed point number. From [2] however, it is known that 16-bit fixed point samples suffer large quantization noise, especially while 64-QAM modulation is applied.

3.3.2 Synchronization

A HiperLAN/2 burst is always preceded by a known sequence of special OFDM symbols, which is called preamble. This preamble can be used to detect the beginning of an OFDM burst. The synchronization function

signals the controller of the receiver in which operation state the receiver is. Different states are defined: START, PREAMBLE A, PREAMBLE B, PREAMBLE C and DATA. These states are indicated by different preambles: PREAMBLE A, PREAMBLE B SHORT, PREAMBLE B LONG and PREAMBLE C. The patterns of the preambles are known at the receiver through lookup tables. For each preamble a lookup vector of 64 elements is defined. Two mechanisms are actually implemented in the synchronization function:

- Detection of transmission:
  the average power of the last 16 input samples is determined.
- Detection of different preambles:
  Preamble A and Preamble B are detected using a matched filter of 16 samples. For Preamble A and Preamble B, all 80 samples are used to detect the preamble. Hence 5 peaks are detected after convolution ($5 \times 16 = 80$). Preamble C is detected using a filter of 32 samples. During Preamble C detection only 96 ($3 \times 32 = 96$) samples are used, so only 3 peaks are detected after convolution. The remainder of Preamble C is used for channel estimation by the channel equalizer.

The output from a matched filter with N samples and two complex inputs x and y is

$$output[l] = \frac{1}{N} \sum_{i=1}^{N} x[i]y[l+i]^* \tag{3.1}$$

where y^* denotes the complex conjugate of y.

3.3.3 Prefix removal

The function prefix removal can be divided into several functions:

- Tracking the symbol
- Removing the prefix part from the symbol

The cyclic prefix of a data OFDM symbol contains a copy of the last 16 samples of the useful data part of that symbol. Hence this redundant information can be used to find the beginning of the symbol. Therefore the correlation is calculated between 16 samples and 16 samples received 64 samples earlier.

In order to avoid a waste of calculation power to calculate this correlation for each incoming sample, an estimate is made for the location of the symbol start. During the search for the symbol start, the length of the cyclic buffer is set to 96 samples. This gives the symbol window the opportunity to move eight samples in both directions, since a symbol is only 80 samples long. The following information is provided to the controller of the receiver:

- Start of symbol:
  Index in the cyclic buffer that contains the first sample of the prefix of the OFDM symbol.

- Decode stop:
  Tells the receiver when the next demodulation cycle is due, which is also an estimate for the start of the next symbol.

- Difference:
  The difference between estimate start and found start of symbol. Large differences tell the controller that synchronization failed.

Because the symbol window is 16 samples larger than the actual symbol, the outcome of the matched filter has to be determined sixteen times, each time advancing one sample. A peak in the output of the matched filter indicates the (temporary) start of the symbol. The expected start of the next symbol, or decode stop, is set to 88 samples from the start of the current symbol.

Now the start of the symbol is found, the first 16 samples of the OFDM symbol can be removed. Those 16 samples are also denoted as prefix and contain no useful information for the decoding. The last 64 samples in the useful part of the OFDM symbol are copied to a length 64 cyclic buffer. All functions after the prefix removal function will operate on the new cyclic buffer, since it is a waste of calculation power to correct errors in the prefix.

3.3.4 Frequency offset correction

There exists a difference in mix frequencies between transmitter and receiver, which causes inter-subcarrier interference. Using information that can be retrieved from the received preamble sections, the receiver can compensate for frequency offsets. The frequency offset is determined in the time domain. In the time domain a frequency offset causes a phase-shift. Hence the frequency offset can be determined by taking the phase-shift between a known and a received signal.

$$\Theta(a,b) = \arctan\left(\frac{\Re(a)}{\Im(a)}\right) - \arctan\left(\frac{\Re(b)}{\Im(b)}\right) \tag{3.2}$$

The angle between the input samples and known Preamble A is used to determine the frequency offset, as shown in equation 3.2. For almost the entire preamble section, the average rotation per sample is calculated according to the regression line. Only 64 samples of the preamble section are utilized for determining the rotation per sample.

The frequency offset is corrected by calculating the current rotation of a sample. The complex samples are then rotated with a negative amount of

that rotation similar to equation 3.3.

$$\begin{aligned} \Re(x_{corrected}) &= \Re(x)\cos(\phi) - \Im(x)\sin(\phi) \\ \Im(x_{corrected}) &= \Im(x)\cos(\phi) - \Re(x)\sin(\phi) \end{aligned} \tag{3.3}$$

3.3.5 Inverse orthogonal frequency division multiplexing

64 time domain samples represent the useful data part of the OFDM symbol that has to be demodulated. Before demodulation can take place, the subcarrier values must be retrieved from the useful data part. This can be done by applying a fast Fourier transform (FFT) to the vector containing the 64 samples. The FFT efficiently implements a discrete Fourier transform (DFT), given in equation 3.4.

$$\hat{f}_n[x] = \sum_{m=0}^{N-1} \tilde{s}_n[m] e^{-j2\pi \frac{xm}{N}} \tag{3.4}$$

with $x = 0, ..., 63$ and

$$\hat{f}_n = \left[0\ \hat{C}_{n,1}\ \hat{C}_{n,2}\ ...\ \hat{C}_{n,26}\ 00000000000\ \hat{C}_{n,-26}\ ...\ \hat{C}_{n,-2}\ \hat{C}_{n,-1}\right]$$

From this vector the 52 subcarrier values can be extracted. In the implementation of the receiver [2], an in-place FFT has been implemented with the advantage that the same vector is used for both input and output.

3.3.6 Common phase offset correction

Common phase offset occurs when mixers in transmitter and receiver do not have the same phase at a given time, under assumption that there is no frequency offset. The common rotation of the subcarriers can be determined by calculating the mean rotation of the pilot carriers compared to their expected values.

$$Offset = \frac{1}{4}\left(\frac{\hat{C}_{normalized,57}}{PilotValue} + \frac{\hat{C}_{normalized,43}}{PilotValue} + \frac{\hat{C}_{normalized,7}}{PilotValue} + \frac{\hat{C}_{normalized,21}}{PilotValue}\right) \tag{3.5}$$

In equation 3.5 the process of determining the common phase offset is shown. The received subcarrier values $\hat{C}$ of the pilots are normalized and their rotation according to the expected pilot value is determined. normalization is normally performed in the Analog-to-Digital Converter (ADC). The common rotation is the averaged result over all four pilot tones. Once the common phase offset is known, one can apply phase offset correction to all received subcarrier values.

$$\hat{C}_{input,corrected}[l] = \frac{\hat{C}_{input,uncorrected}[l]}{Offset} \tag{3.6}$$

In contrast with the frequency offset correction (equation 3.2 and 3.3), one does not determine the phase offset rotation. Instead of determining the rotation, one determines a complex scaling factor. Advantage of this approach is that there are no $arctan()$, $cos()$ and $sin()$ calculations involved in the common phase corrections.

3.3.7 Channel equalization

The channel estimator in the receiver model uses Preamble C to determine an estimation of the channel. The estimate of the channel is determined by comparing the known Preamble C and the received subcarrier values. The channel is estimated for 64 samples, which equals the length of the useful data part.

$$\frac{1}{\hat{H}_l} = \frac{\overrightarrow{C}_{preamble}[l]}{\hat{f}_n[l]} \tag{3.7}$$

Before de-mapping each subcarrier value is corrected.

$$\hat{C}_\gamma = \frac{\hat{C}_\gamma}{\hat{H}_\gamma} \tag{3.8}$$

3.3.8 De-mapping

In HiperLAN/2 there are four mapping techniques available: BPSK, QPSK, 16-QAM and 64-QAM. Each of these techniques has a different number of bits per complex symbol. By way of a lookup table the output bits are determined. In the lookup table, all possible subcarrier values for a certain mapping scheme are defined. The most likely symbol that was transmitted is probably the symbol to which the Euclidian distance in the lookup table is smallest. In equation 3.9 the definition of the Euclidean distance is given.

$$|a - b| \equiv \sqrt{\left(\Re(a) - \Re(b)\right)^2 + \left(\Im(a) - \Im(b)\right)^2} \tag{3.9}$$

To find the nearest subcarrier value to the received subcarrier value, one compares the received subcarrier value with all subcarrier values in the lookup table. To each subcarrier value in the lookup table a certain bit pattern is associated.

For BPSK 2 subcarrier values are stored in the lookup table, for QPSK, 16-QAM and 64-QAM there are stored 4, 16 and 64 subcarrier values, respectively. This method of de-mapping is called hard decision de-mapping.

3.4 Computational requirements

In this section we will give the computational requirements in order to execute the functions that are given before. We consider the MONTIUM archi-

tecture [4] and assume that the clock frequency of the MONTIUM-tiles is 100 MHz. So one clock cycle has a duration of 10 ns.

3.4.1 Lookup tables

First of all we will look to the size of the lookup tables, which are necessary for the synchronization, frequency offset correction and de-mapping functions.

Information in lookup table	Entries in lookup table
Preamble A	64 complex numbers
Preamble B	64 complex numbers
Preamble C	64 complex numbers
Carrier values BPSK	2 complex numbers
Carrier values QPSK	4 complex numbers
Carrier values 16-QAM	16 complex numbers
Carrier values 64-QAM	64 complex numbers

Table 3.1: Size of information stored in lookup tables

In table 3.1 it is seen that a large lookup table has to be present. Most lookup tables use 64 complex entries, hence 128 positions in the lookup table are actually in use. However, the lookup table can be split in a table containing the real data-part and one containing the imaginary data-part.

3.4.2 Matched filters

In the synchronization and prefix removal functions the preambles are detected with matched filters. In table 3.2 the sizes of the matched filters that are applied in the different functions are shown.

Detection of	Size of matched filter
Preamble A	16 complex samples
Preamble B	16 complex samples
Preamble C	32 complex samples
Prefix	16 complex samples

Table 3.2: Sizes of the matched filters that are applied in different functions

Because the window is 16 samples larger than the actual symbol, the outcome of the matched filter during prefix-detection has to be determined 16 times, each time advancing one sample. A peak in the consecutive outputs of the matched filter indicates the start of the OFDM symbol. During the detection of the prefix a cyclic buffer of length 96 samples is used. After the

$\underline{\text{ALU}-1}$
$A \coloneqq a[i]$
$C \coloneqq c[l+i]$
$D \coloneqq sum_{local,real}$
$Z2 \coloneqq A * C + EAST$
$Z3B \coloneqq Z2 + D$
$OUT \coloneqq Z3B$
$\underline{\text{ALU}-2}$
$A \coloneqq b[i]$
$C \coloneqq d[l+i]$
$WEST \coloneqq A * C$

Figure 3.2: Calculation of the real part of the cross-correlation with 2 ALUs in the MONTIUM

$\underline{\text{ALU}-3}$
$A \coloneqq b[i]$
$C \coloneqq c[l+i]$
$D \coloneqq sum_{local,imaginary}$
$Z2 \coloneqq A * C + EAST$
$Z3B \coloneqq Z2 + D$
$OUT \coloneqq Z3B$
$\underline{\text{ALU}-4}$
$A \coloneqq a[i]$
$C \coloneqq d[l+i]$
$WEST \coloneqq A * C$

Figure 3.3: Calculation of the imaginary part of the cross-correlation with 2 ALUs in the MONTIUM

prefix is detected, the useful part of the OFDM symbol is copied to a cyclic buffer of length 64.

In equation 3.1 the cross-correlation between two complex vectors is defined. During matched-filtering the cross-correlation is determined in order to detect preambles and prefixes. One possible way to implement the cross-correlation function in the MONTIUM-structure is a separate approach of calculating the real-part and the imaginary-part of the correlation, which is shown in figure 3.2 and 3.3.

We assume $x[i] = a[i] + jb[i]$ and $y[i] = c[i] + jd[i]$. So with 4 ALUs one can determine the real and imaginary part of the complex cross-correlation. In order to result in one valuable output, one has to determine the absolute value of the complex cross-correlation and also a scaling factor $1/N$ has to be applied. Since square root calculations are computationally complex, we will determine the squared absolute value of the cross-correlation instead of the absolute value. Hence the square root calculation is avoided. Scaling with

$1/N$ is also neglected, since it only scales the output of the cross-correlation calculation. However, the shape of the correlation function will remain the same, so peaks can still be detected.

Considering the complex cross-correlation one can conclude that 4 ALUs are utilized in order to perform a cross-correlation calculation of one sample. When the complex cross-correlation of N samples has to be calculated, this calculation will spend $N + 1$ clock cycles. The real part and the imaginary part of the sum can be calculated in parallel using 4 ALUs. When the entire sum for all N samples is calculated, one has to change the configuration of the ALUs in order to calculate the squared absolute value of the complex correlation. This operation will absorb an extra clock cycle. In total 2 different ALU configurations are used to perform the complex correlation. A summary of the requirements for complex cross-correlation is given in table 3.3.

# clock cycles	$N + 1$
# ALU configurations	2
# tiles	1

Table 3.3: Requirements for a complex correlation of N samples on the MONTIUM architecture

3.4.3 Frequency offset correction

The estimation function of the frequency offset utilizes 64 samples of Preamble A to determine the average rotation per sample. Hence the phase shift between the received sample and the known preamble sample has to be determined 64 times. When the average rotation per sample is determined, each sample can be corrected with a certain phase-rotation. Disadvantage of this approach is that $arctan()$, $sin()$ and $cos()$ calculations are applied.

The regression slope is defined as:

$$\frac{N \sum_{i=0}^{N-1} x[i]y[i] - \sum_{i=0}^{N-1} x[i] \sum_{i=0}^{N-1} y[i]}{N \sum_{i=0}^{N-1} x^2[i] - \left(\sum_{i=0}^{N-1} x[i] \right)^2}$$

Herein $y[i]$ is the measured rotation between the received sample and the expected preamble value, $\Theta\left(\overrightarrow{v}_{preamble}[i], \overrightarrow{v}_{input}[i]\right)$. $x[i]$ is the sample index and N is the number of samples that are used. The equation can be simpflified to:

$$\frac{64\sum_{i=0}^{63} i\Theta\left(\overrightarrow{v}_{preamble}[i], \overrightarrow{v}_{input}[i]\right) - 2016\sum_{i=0}^{63}\Theta\left(\overrightarrow{v}_{preamble}[i], \overrightarrow{v}_{input}[i]\right)}{1397760}$$

Not the entire preamble section is used for estimating the frequency offset. The first 16 samples of the preamble are neglected, while they can contain inter-preamble interference.

One should investigate if the complex divisions can be performed with lookup tables. Furthermore one has to know how these lookup tables can be used. We have to compare lookup tables that are initialized with inverse complex numbers or with angle information. In this way we assume a complex division as a multiplication with the inverse of the complex number.

3.4.4 Fast Fourier Transform

The inverse orthogonal frequency division multiplexing is just a 64-FFT operation. This operation is performed by radix-2 butterflies and consumes $64\log_2(64)$ complex multiplications. On the MONTIUM structure the 64-FFT can be performed in $\left(\frac{64}{2}+1\right)\log_2(64)$ clock cycles.

$\log_2(64)+4$ different memory configurations are used to perform the 64-FFT on the MONTIUM and 8 different crossbar configurations have to be used. There is only 1 ALU configuration required, since for each operation the same radix-2 structure (butterfly) is used.

The FFT algorithm can be performed with 4 ALUs and 10 memory banks, which corresponds to one MONTIUM-tile, as given in table 3.4. From [4] we know that there can exist 64 possible memory configurations and 64 possible crossbar configurations in one design.

# clock cycles	198
# crossbar configurations	8
# memory configurations	10
# ALU configurations	1
# tiles	1

Table 3.4: Requirements for 64-FFT on the MONTIUM architecture

3.4.5 Common phase offset correction

In order to estimate the common phase offset, one determines the mean rotation of the pilot carriers compared to their expected values. For every OFDM symbol the expected pilot values can be calculated. The pilot sequence can be created with the generation polynomial:

$$X_7 \oplus X_4 \oplus 1$$

$\oplus$ denotes a modulo two adder, which is also known as an "exclusive or". $X_{1...7}$ represents the state of the scrambler.

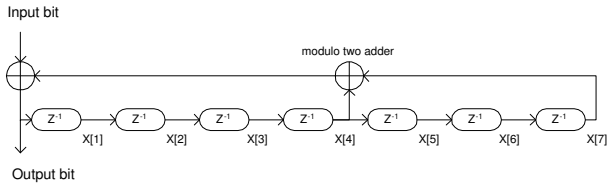

Figure 3.4: Scrambler of HiperLAN/2 physical layer

The scrambler should be initialised with all 1's and a 1 in the output should be replaced by '-1' and a 0 should be replaced with '1'. The replacement of 1 and 0 can be characterised by a function:

$$pilot = -2 \times output + 1$$

During each OFDM symbol, the internal state of the scrambler has to be stored in memory, which requires 7 entries in the memory. It is certainly preferred to perform the pilot generation in a FPGA, because only single bits are involved in the process instead of complete words.

For each received pilot value the rotation is derived by dividing the normalized complex received value by the expected value. Where the normalization of a complex value is defined as:

$$z = a + jb \Longleftrightarrow z_{norm} = \frac{z}{|z|} = \frac{a + jb}{\sqrt{a^2 + b^2}}$$

Normally the normalization is performed by the ADC, which translates the signal value from the analog to the digital domain.

The expected value can be 1 or -1. The average of all four rotations is determined and used for correction. Now every of the 64 complex values is compensated, whereby every value is divided by a scaling factor (as seen in equation 3.6). Just like frequency offset correction, one has to deal with complex divisions. In table 3.5 a summary of the requirements is given.

3.4.6 Channel equalization

The channel equalization function determines the inverse channel transfer of every sample independently. Each complex value of the received Preamble C is divided by the corresponding expected Preamble C value, which results in a scaling factor. So the scaling factor is different for each index. Each

Common phase offset	# complex divisions
normalization of input samples	4
Phase offset estimation per OFDM symbol	4
Phase offset correction per OFDM symbol	64

Table 3.5: Computational requirements for phase offset correction in HiperLAN/2

complex value of the useful part of the OFDM value is corrected with the scaling factor that corresponds to its index.

For each OFDM burst one has to calculate 64 scaling factors, which is in fact the inverse transfer function of the channel. Furthermore each complex value of the useful part of the OFDM symbol has to be divided by the scaling factor, which requires 64 complex divisions.

Channel equalization	# complex divisions
Channel estimation per burst	64
Channel correction per OFDM symbol	64

Table 3.6: Computational requirements for channel equalization in HiperLAN/2

3.4.7 De-mapping

The de-mapping in the model is implemented as hard decision de-mapping. In HiperLAN/2 there are four modulation schemes available: BPSK, QPSK, 16-QAM and 64-QAM. The hard output bits are determined by comparing the soft output values with values in a lookup table. In table 3.1 the size of the lookup table for the different modulation schemes is given. The most likely symbol that was transmitted is probably the symbol to which the Euclidean distance in the lookup table is smallest.

Depending on the used modulation scheme, one has to determine the Euclidean distance 2, 4, 16 or 64 times for BPSK, QPSK, 16-QAM or 64-QAM, respectively. From these Euclidean distances one has to determine the minimum value (table 3.7). The hard output bits are equal to the index of the minimum value in the lookup table (when the lookup table is well initialised).

The consumption time of the de-mapping function can however be reduced, while one first determines in which quadrant the received soft-value is situated. After that one has only to determine the Euclidean distances in that corresponding quadrant. Hence, the time during calculation is reduced by a factor 4 as seen in table 3.8.

Modulation scheme	# Euclidean distance calculations	# minimum calculations
BPSK	2	2
QPSK	4	4
16-QAM	16	16
64-QAM	64	64

Table 3.7: Computational requirements for full de-mapping in HiperLAN/2

Modulation scheme	# Euclidean distance calculations	# minimum calculations
BPSK	1	1
QPSK	1	1
16-QAM	4	4
64-QAM	16	16

Table 3.8: Computational requirements for (reduced) de-mapping in HiperLAN/2

In equation 3.9 the definition for calculating the Euclidean distance is given. Calculating the Euclidean distance requires a square root calculation. Since the Euclidean distance is only calculated for determining the minimum Euclidean distance, it is also allowed to determine the minimum value of the squared Euclidean distance. In this way the square root calculation is overruled.

Using 2 ALUs one can calculate the squared Euclidean distance between the known complex value and the soft output of the FFT function. When all these distances are determined, one has to determine a minimum distance, which can be performed with 1 ALU that is comparing all distances in a certain amount of clock cycles. However, the translation from the minimum distance to the memory address will generate heavy problems. Hence, the memory address corresponds to the hard output bits.

During one clock cycle one can perform two Euclidean distance calculations using one tile at the Montium-architecture.

In figures 3.5 and 3.6 a possible mapping of finding the smallest local, squared Euclidean distance is shown. Within one Montium-tile one can determine two squared Euclidean distances in parallel using 4 ALUs. The fifth ALU in the Montium-tile can be used to determine the minimum of the calculated distances. First the minimum from the outputs of ALU-1 and ALU-3 is determined. Hereafter this obtained minimum is compared to the local minimum, which is obtained in foregoing clock cycles. Using the mapping, given above, one can perform the searching for minimum Eu-

$$
\begin{aligned}
&\underline{\text{ALU}-1}\\
&f_1 : z_1 := \Re(a) - \Re(b)\\
&f_2 : z_2 := z_1\\
&f_3 : z_3 := z_1\\
&OUT := z_2 * z_3 + EAST\\
&\underline{\text{ALU}-2}\\
&f_1 : z_1 := \Im(a) - \Im(b)\\
&f_2 : z_2 := z_1\\
&f_3 : z_3 := z_1\\
&WEST := z_2 * z_3
\end{aligned}
$$

Figure 3.5: Calculation of the squared Euclidean distance with 2 ALUs in the MONTIUM

$$
\begin{aligned}
&\underline{\text{ALU}-1}\\
&OUT := d_1\\
&\underline{\text{ALU}-3}\\
&OUT := d_2\\
&\underline{\text{ALU}-5}\\
&f_1 : z_1 := \min(d_1, d_2)\\
&f_2 : z_2 := d_{\text{local\_minimum}}\\
&f_3 : z_3 := \min(z_1, z_2)\\
&OUT := z_3
\end{aligned}
$$

Figure 3.6: Determination of the minimum Euclidean distance within 1 MONTIUM-tile

clidean distance in 1, 2, 8 or 32 clock cycles for BPSK, QPSK, 16-QAM or 64-QAM, respectively. When the quadrant of the complex plane is considered, the computation time can be reduced by a factor 4. The requirements for de-mapping one complex value while considering the quadrant in the complex plane is shown in table 3.9. Since each OFDM symbol consists of 48 complex values, the computation time of the entire OFDM symbol is 48 times larger than the numbers given above.

Modulation scheme	**# clock cycles**
BPSK	1
QPSK	1
16-QAM	2
64-QAM	8

Table 3.9: Computational requirements for de-mapping one complex value in HiperLAN/2

3.5 Clustering receiver algorithms

Mapping the HiperLAN/2 receiver algorithms on the MONTIUM architecture requires knowledge of the typical HiperLAN/2 symbol durations. In section 3.4 we have discovered the typical computational requirements of these algorithms. Furthermore we discovered that all algorithms can be mapped on one tile. In table 3.10 we will summarize the time consumption of all the algorithms, while they are performed on a single tile and the number of ALU configurations that are utilized.

Functional block	**# clock cycles**	**# ALU configurations**	**Computation time @ 100 MHz [μs]**
Prefix removal	272	2	2.72
Frequency offset correction	64	2?	0.64
Inverse OFDM	198	1	1.98
Phase offset correction	128	2?	1.28
Channel equalization	64	2?	0.64
De-mapping	48 – 384	1	0.48 – 3.84
synchronization and control	?	?	?

Table 3.10: Computational requirements of HiperLAN/2 receiver algorithms mapped on the MONTIUM architecture

One major property of signaling in HiperLAN/2 physical layer is the aspect of OFDM symbols. All operations in the physical layer are performed

on these OFDM symbols. An OFDM symbol has a fixed length of 80 samples. Hence, at a sample rate of 20 MHz the duration corresponds to 4 μs. One should assure that each 4 μs a new OFDM symbol can be processed.

In the foregoing part we already discovered that the receiver algorithms need 2 ALU configurations at most, when performed on a single tile. The complete receiver processing would require about 11 μs excluding the synchronization and control processing. Hence, one OFDM symbol can not completely be processed by all receiver algorithms within its duration of 4 μs. So scheduling of tasks on different tiles has to be performed. From the table can be seen that only 10 ALU configurations are required for performing all receiver algorithms. Based on the information of the MONTIUM architecture [4], the complete HiperLAN/2 receiver can be mapped on a single tile; up to 64 ALU configurations can be stored in the ALU configuration register. Because of timing constraints, while each 4 μs an OFDM symbol has to be processed, 3 MONTIUM-tiles are required in order to perform all receiver processing. Different tasks have to be scheduled over these 3 tiles. Extra tiles have to be used in order to perform the synchronization and control part of the receiver.

3.6 Conclusion and discussion

We have analysed the receiver part of the HiperLAN/2 physical layer. Different functions in the receiver are considered. The description in this chapter is all based on a model. For each function we tried to describe the characteristics that are important while mapping onto reconfigurable hardware. During the mapping of the functions, we have only considered the MONTIUM architecture. For each function we tried to give some general rules in order to map that function onto the architecture.

There are some foreseen problems, mostly with square root calculations that are involved in the magnitude calculations of complex values. Moreover the mapping of divisions can be difficult. Especially the frequency offset correction and phase offset correction could suffer from this, so maybe these functions could be better performed in General purpose processors completely or partly. Another possibility of performing (complex) divisions is utilization of lookup-table functionality in the MONTIUM-tile.

Chapter 4

Bluetooth physiscal layer

4.1 Introduction

In this chapter we will study the Bluetooth communication system. We will discover the typical functional blocks in the communication system and investigate the requirements for implementing the receiver of this system in reconfigurable hardware. The discussion in this paper will be partly based on the results of the *SDR* project of the University of Twente [3], furthermore the design approach of the *Chameleon* project will be used [1].

In section 4.2 the behaviour of the Bluetooth physical layer will be discussed. The Bluetooth protocol stack will be shown and for the lower relevant layers the most relevant properties will be shown. We are mostly interested in the Radio processing layer, but to understand its functioning one also has to known the most important properties of the higher layers. Therefore the Baseband layer is also discovered. In section 4.3 the Bluetooth communication model is shown with all its processing mechanisms in the receiver. Once we know which functions have to be performed, we will go into detail and focus on the computational requirements of these receiver functions in section 4.4. The implementation of an Bluetooth receiver in the MONTIUM architecture is studied in section 4.5. Based on the computational requirements, like sample rate, possible implementations show that a Bluetooth receiver can be implemented on a MONTIUM-tile. Finally the achieved results, like processing delay and number of ALU configurations, are discussed in section 4.6. In section 4.7 a concluding summary is given.

4.2 Physical layer in Bluetooth

The task of the physical layer in Bluetooth is to modulate bits that origin from the data layer on the transmitter side and to demodulate them on the receiver side, and vice versa.

The Bluetooth system consists of a radio unit, a link control unit and

a support unit for link management and host terminal interface functions (figure 4.1). All radio processing is performed in the radio unit. The link controller carries out the baseband protocols and low-level link routines. Link layer messages for link set-up and control are handled by the link manager.

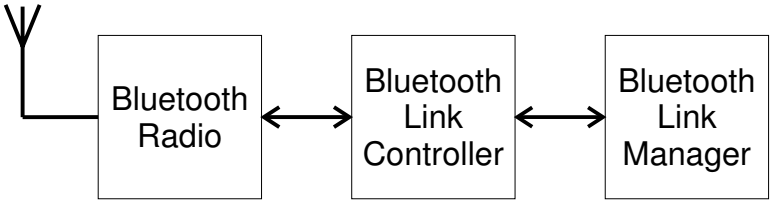

Figure 4.1: Functional blocks in the Bluetooth system

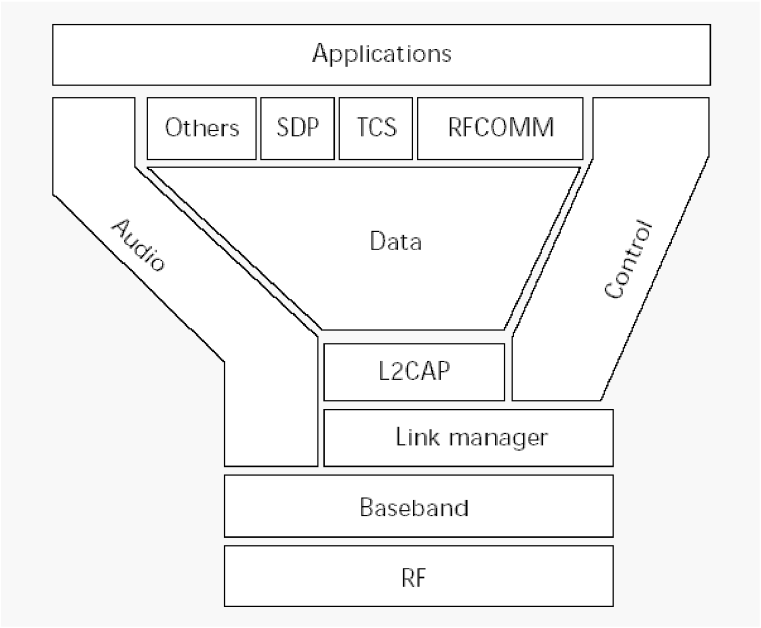

Figure 4.2: The Bluetooth protocol stack

In figure 4.2 the Bluetooth protocol stack is shown. We are especially interested in the two lower layers:

- The RF layer, specifying the radio parameters:
  - air interface
  - modulation
- The Baseband layer, specifying the lower-level operations at the bit and packet levels:

- speech coding
- HEC-, FEC-operations
- CRC calculations
- scrambling, descrambling
- encryption, decryption
- sequencing of frequency hopping

4.2.1 RF layer

The frequency spectrum available to Bluetooth is positioned in an unlicensed radio band that is globally available. This band, the Industrial, Scientific, Medical (ISM) band, is centered around 2.45 GHz. In most countries of the world[1], free spectrum is available from 2400 MHz to 2483.5 MHz. The frequency spectrum is divided into 79 so called channels, which are referred as radio channels. Each of those radio channels occupies a bandwidth of 1 MHz.

In the ISM band, the signal bandwidth of the Bluetooth system is limited to 1 MHz. For robustness, a binary modulation scheme was chosen. With the mentioned bandwidth restriction, the data rates are limited to about 1 Mbps. Bluetooth uses Gaussian-shaped frequency shift keying (GFSK) modulation with a nominal modulation index of $k = 0.32$. Logical ones are sent as positive frequency deviations, logical zeros as negative frequency deviations. Demodulation can simply be accomplished by a limiting FM discriminator. This modulation scheme allows the implementation of low-cost radio units.

4.2.2 Baseband layer

Frequency Hopping Spread Spectrum

Bluetooth is based on Frequency-hopping(FH)-CDMA, which is also known as Frequency Hopping Spread Spectrum. In the 2.45 GHz ISM band, a set of 79 hop carriers has been defined at a 1 MHz spacing[2]. The channel is a hopping channel with a nominal hop dwell time of 625 μs. In the time domain, the channel is divided into slots. The minimum dwell time of 625 μs corresponds to a single slot. To simplify implementation, full-duplex communication is achieved by applying time-division duplexing (TDD). This mean that a unit alternately transmits and receives data. The transmission and reception of data takes place at different hop carriers. The master starts data transmission only in even numbered slots, while slaves start transmitting in odd numbered slots.

[1] In France and Spain the bandwidth of the free spectrum is smaller.

[2] Currently, for France and Spain a reduced set of 23 hop carriers has been defined [5].

Packet-based communications

The Bluetooth system uses packet-based transmission: the information stream is fragmented into packets. In each slot, only a single packet can be sent. All packets have the same format, starting with an access code, followed by a packet header, and ending with the user payload.

There are different types of packets that are used to define packets for synchronous and asynchronous services, which are divided in segments:

- Segment 1, specifies packets that fit into a single slot
- Segment 2, specifies 3-slot packets
- Segment 3, specifies 5-slot packets

Multi-slot packets are sent on a single-hop carrier. The hop carrier that is valid in the first slot is used for the remainder of the packet; therefore there is no frequency switch in the middle of the packet. After the packet has been sent, the hop carrier as specified by the current master clock value is used, as seen in figure 4.3. Note that only an odd number of multi-slot packets has been defined, which guarantees that the TX/RX timing is maintained.

The Bluetooth link supports both synchronous communication, such as voice traffic, and asynchronous communication, such as bursty data traffic. Two physical link types have been defined:

- Synchronous connection-oriented (SCO) link
- Asynchronous connection-less (ACL) link

The SCO link is a point-to-point link between the master and a single slave. The link is established by reservation of duplex slots at regular intervals. The ACL link is a point-to-multi-point link between the master and all the slaves. The slotted structure of the radio channels allows effective mixing of synchronous and asynchronous links.

4.3 Bluetooth communication model

The baseband communication in the Bluetooth system is very simple. The data to be transmitted is stored in a buffer, and in the radio interface the Baseband processing and radio processing are done. In the receiver similar processing has to be done, radio processing and Baseband processing are done and afterward the information is stored in a buffer. So the RF layer and the Baseband layer are covered by the radio interface. In for example [6] is seen that in a Software Defined Radio system the Radio Frequency (RF) processing is performed in analog hardware. Modulation and channel filtering is however performed in the digital domain. Channel selection is

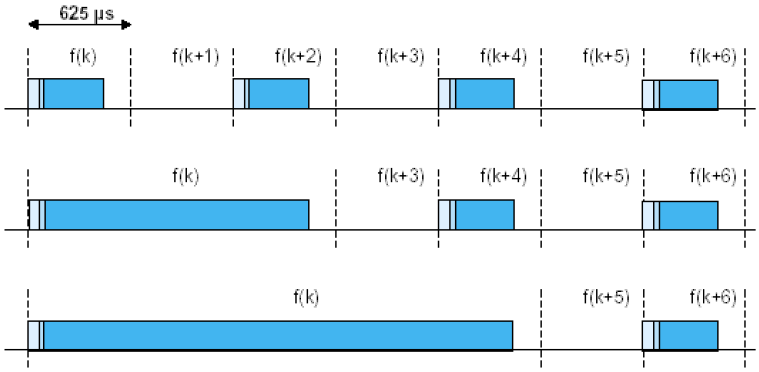

Figure 4.3: The frequency and timing characteristics of single-slot, three-slot, and five-slot packets

not yet discussed in this chapter, however it should be implemented since it conveys the aspects adaptability and reconfigurability in SDR environments [7].

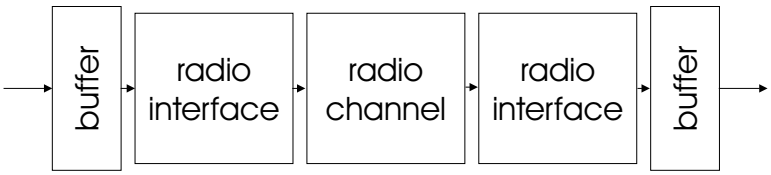

Figure 4.4: The Bluetooth communication model

4.3.1 Bluetooth transmitter

In the transmitter part, the binary signal is pulse-shaped by using a Gaussian filter. The train of Gaussian pulses is afterward Frequency Modulated. The resulting transmitted signal conveys all its information in the frequency deviation of the signal.

4.3.2 Bluetooth receiver

In the receiver part, the transmitted radio signal is converted back into a binary NRZ signal. The radio signal, which is received, conveys all its information in the frequency deviation of the signal. One possible way to demodulate a FM signal is by way of FM-to-AM conversion, which is also called a FM-discriminator. The FM-discriminator allows the implementation for low-cost radio units, which is essential for Bluetooth systems. In the FM-discriminator, which is shown in figure 4.5, the received signal is multiplied with its delayed version. After FM-to-AM conversion, the signal

is passed through a Low Pass Filter and finally the consecutive bits are detected by a threshold detector. The threshold detector is not shown in figure 4.5. The Low Pass Filter is just a FIR (Finite Impulse Response) filter. The FIR filter applies multiply and accumulate operations as seen in figure 4.6.

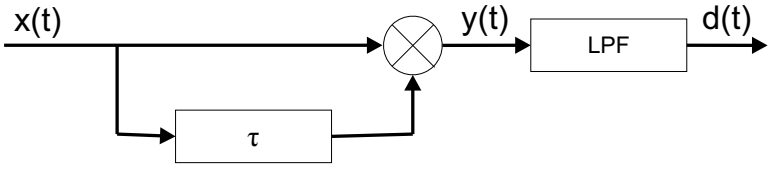

Figure 4.5: Block diagram of the FM-discriminator

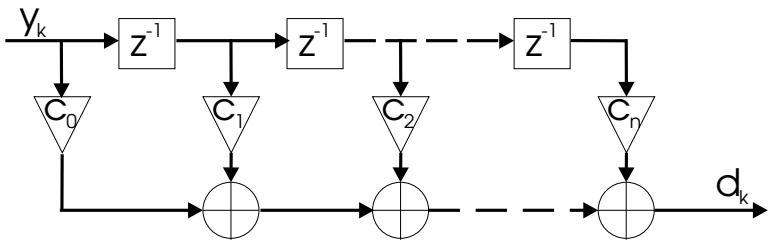

Figure 4.6: Finite Impulse Response filter

4.4 Computational requirements

In this section we will give the computational requirements in order to execute the functions in the receiver of the RF layer and Baseband layer. The requirements are partially extracted from the results of the *SDR* project [3].

4.4.1 Receiver

The operations in the FM-receiver can be divided into three processes. The operations are performed on a Bluetooth signal ($x(t)$ in figure 4.5) with a sample rate of 10 MSPS [7]:

- a multiplication process, wherein the Bluetooth signal is multiplied with a delayed version. The signal has to be delayed $\tau = \frac{1}{4f_c}$ seconds [8]. When the center frequency, f_c, of the Bluetooth signal is 2.5 GHz, one has to set the delay, τ, to 100 ns. The sample rate of the Bluetooth signal in the SDR testbed is equivalent to 10 MSPS. Consequently, the signal has to be delayed by 1 sample time.

- a multiply and accumulate process, wherein FIR filtering is performed. The FIR filter is applied in order to pass the slowly varying AM signal with a frequency of 1 MHz and to block all high frequencies that occur due to multiplication. Actually the FIR filter, as depicted in figure 4.5, operates at a sampling rate of 10 MSPS, however it is sufficient to operate at a sample rate of 1 MSPS.

- a decision process, wherein decisions about the received bit value are taken. For the threshold detector it is sufficient to operate at a sample rate of 1 MSPS, since the bit rate of the Bluetooth communication system is 1 Mbps. As a consequence, the threshold detector has to perform its bit decisions dependent on only 1 sample per bit, while at a sample rate of 10 MSPS 10 samples can be used to make a decision of bit value.

Oversampling of the Bluetooth signal, $x(t)$ in figure 4.5, has only been done to obtain an accurate delay for the first multiplication process. After that process, the higher sample rate is not necessary anymore and decimation of the sample rate could be applied. The sample rate of the signals after multiplication, $y(t)$ and $d(t)$, could be reduced by a factor 10, since the bit rate of the Bluetooth system is 1 Mbps. However, decimation of the Bluetooth signal, $y(t)$ or $d(t)$ yields large consequences:

- Decimation of $y(t)$
  yields less multiply and accumulate operations for the FIR filter in a certain time interval. When decimation of $y(t)$ is performed, the real-time performance in terms of time consumption improves. After decimation by a factor 10, one can increase the number of filter coefficients by a factor 10 and consequently the computation time remains the same.

- Decimation of $d(t)$
  yields less samples per bit in order to make a decision about the bit value. The FIR filtering is performed at the oversampled rate, for real-time performance however, less filter coefficients can be applied in the same real-time computation interval.

4.5 Implementation

We will now focus on the implementation of the receiver functions in a Montium architecture [4]. However, the Bluetooth receiver can be implemented in other embedded architectures, like FPGA or General Purpose Processors, as well.

4.5.1 FM-discriminator

In the FM-discriminator (figure 4.5) the incoming FM signal is multiplied with its delayed version. Since the incoming signal is sampled at a frequency of 10 MHz, the signal only has to be delayed with one sample time under the assumption that the center frequency of the incoming signal is 2.5 GHz.

In one MONTIUM-tile one can perform 5 multiplications in parallel. An advantage of the Bluetooth FM signals is that all sample values represent real numbers. Consequently, 5 real multiplications can be calculated in one clock-cycle.

The processing time of the FM-discriminator depends on the amount of samples that has to be processed at the initialization phase; the FIR filter has to be initialized with an amount of samples that is equal to the number of taps.

Suppose that a FIR filter with 10 taps is employed, so 10 sample values have to be generated by the FM-discriminator, before the FIR filter is initialized. Since the incoming FM modulated signal has to be delayed with one sample time, 10 + 1 samples have to be loaded in the local memory before the signal processing can start.

4.5.2 FIR filter

The length of the FIR filter, used in the FM-discriminator, is not variable. However, in a MONTIUM-tile one can perform a FIR filter with a maximum of 2560 taps. In all ALUs multiply and accumulate operations are performed. In figure 4.7 and 4.8 possible mappings of a FIR filter onto a MONTIUM-tile are shown. Both implementations of the FIR filter yield identical results, but the computation delay of both alternatives differs. Utilizing the EAST-WEST interconnect between the ALUs gains 1 clock cycle while performing FIR filtering. However, the impact of utilizing the EAST-WEST interconnect is not known, but it is suggested that the interconnect can affect the overall clock frequency of the MONTIUM-tile.

When there are no EAST-WEST interconnects applied, each ALU has to store its temporary result (sum_x). When the multiply and (partially) accumulate process is finished, the temporary results of all ALUs have to be collected and summed. This addition of the temporary accumulated results requires one extra clock cycle. Depending on the implementation without or with EAST-WEST interconnect, FIR filtering with N coefficients can be performed in $\frac{N}{5} + 2$ or $\frac{N}{5} + 1$ clock cycles, respectively.

Both implementations of the FIR filter can be applied in 'streaming mode' or in 'block mode'.

- In 'streaming mode' the FIR filtering is applied onto an information flow. The coefficients of the FIR filter are stored in memory, and also

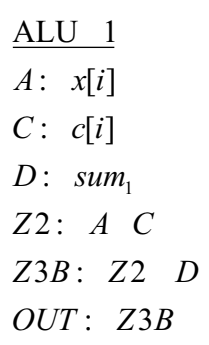

Figure 4.7: FIR filter implementation with 5 ALUs and without EAST-WEST interconnect in the MONTIUM

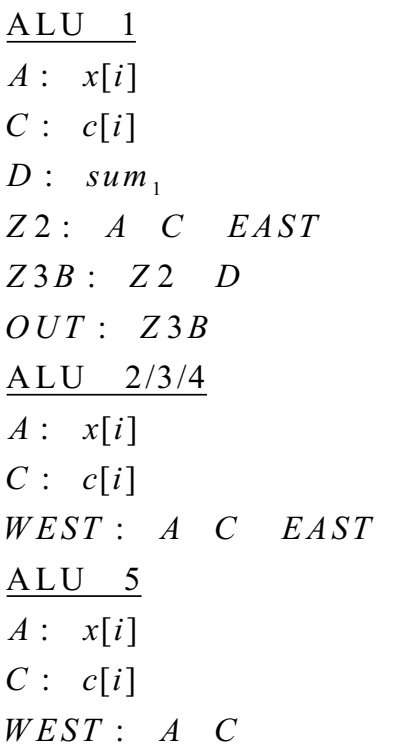

Figure 4.8: FIR filter implementation with 5 ALUs and with EAST-WEST interconnect in the MONTIUM

the corresponding input samples of the information flow are stored in memory.

- In 'block mode' the FIR filtering is applied onto a block of information samples, this block can have a maximum length of $5 \times 512 = 2560$ values.

4.5.3 Threshold detector

In the threshold detector one has to decide if a '0' or '1' is received. The output of the FIR filter depends on the frequency deviation in the received Bluetooth signal. The signal at the output of the FIR filter varies between -1 and $+1$. This signal has to be translated into 'received' bits, by applying these rules:

$$\text{if } x > 0 \text{ then bit:=1}$$
$$\text{if } x \leq 0 \text{ then bit:=0}$$

These rules are implemented in the MONTIUM architecture using the code in figure 4.9. A decision of a bit value is performed using one ALU and takes only one clock cycle. The actual bit output is determined by the status information of the function unit.

$$\underline{\text{ALU}\quad 1}$$
$$A: \quad d[i]$$
$$f_1 : z_1 : \quad A$$

Figure 4.9: Threshold detector implementation with 1 ALU in the MONTIUM

4.6 Clustering receiver algorithms

Mapping the Bluetooth receiver algorithms on the MONTIUM architecture requires knowledge about the typical Bluetooth symbol timing. In section 4.2.2 we have seen that there are defined single slot, 3-slot and 5-slot packets, which have a duration of 625 μs, 1.875 ms and 3.125 ms, respectively. Actually the slots are not completely occupied by the packets. A single slot packet has a length of 366 bits at maximum, and therefore has a duration of 366 μs. The length of a 3-slot packet is at most 1626 bits, which corresponds to 1.626 ms, and for a 5-slot packet the length is 2870 bits at maximum, which is equal to a duration of 2.87 ms.

From these values of the packet lengths, one can conclude that it should be useful to perform the receiver functions in 'streaming' mode. This means that the processing of the receiver functions starts immediately, and does not wait till a complete packet has been loaded into the local memory. Especially when the incoming Bluetooth signal is oversampled, one has to process large amounts of samples, which requires a large initialization phase in order to load all data in the local memory of the processors. Secondly, a 10 times oversampled 5-slot packet requires 28700 memory positions to store all bits. As a consequence at least 6 MONTIUM-tiles have to be used in order to store the complete 5-slot packet in memory. This approach will be extremely inefficient in multiple ways.

In table 4.1 we will summarize the time consumption of all the algorithms, while they are performed on a single tile and the number of ALU configurations that are utilized. The processing delays in the table are given while processing is done for a single sample. During initialization of the receiver one has to load $N + 1$ samples in the local memory before the processing can start. This initialization phase will consume $ceil\{\frac{N}{5}\} + 1$ clock cycles extra (the ceiling value function, $ceil\{x\}$, computes the smallest

integer not less than x).

Functional block	# clock cycles	# ALU configurations	Computation time @ 100 MHz [ns]
FM-discriminator	1	1	10
N-taps FIR filter	$ceil\{\frac{N}{5}\}+2$	2	$ceil\{\frac{N}{5}\}\times 10+20$
	$ceil\{\frac{N}{5}\}+1$	2	$ceil\{\frac{N}{5}\}\times 10+10$
Threshold detector	1	1	10

Table 4.1: Computational requirements of Bluetooth receiver algorithms mapped on the MONTIUM architecture

4.7 Conclusion and discussion

We have analysed the receiver part of the Bluetooth physical layer. Different functions in the receiver are considered. For each function we tried to describe the characteristics that are important while mapping onto reconfigurable hardware. During the mapping of functions, we have only considered the MONTIUM architecture. For each function we tried to give some general rules in order to map that function onto the architecture.

The Bluetooth receiver showed to be a fairly simple signal processing part, which can be implemented in a MONTIUM architecture quite well. When the signal processing is applied in 'streaming' mode, the computation delays of the different receiver parts seem to be some clock cycles.

During analysis of the Bluetooth receiver we have assumed that channel selection of the right Bluetooth channel was already performed. Since channel selection conveys both adaptivity and reconfigurability, we have to study the channel selection mechanisms in Bluetooth receivers as well. Channel selection is extremely dynamic while for each slot a different subcarrier has to be selected. Consequently, one has to adapt the local oscillator and bandpass filter every 625 μs [7].

Chapter 5

Conclusion and recommendations

5.1 Conclusion

In this report we have analysed the feasibility of implementing communication systems in reconfigurable hardware. In the *Adaptive Wireless Networking (AWGN)* project we will utilize a dynamically reconfigurable heterogeneous architecture for implementation of multiple communication systems. The architecture is heterogeneous is this sense that digital signal processing is performed in general purpose processors, bit-level reconfigurable parts and word-level reconfigurable parts.

Two different communication systems have been examined. A rather complex wireless LAN communication system, the HiperLAN/2 standard, and a less complex communication system, the Bluetooth standard.

The physical layer of the HiperLAN/2 system is discussed and the functionality of an HiperLAN/2 receiver is described using a Software Defined Radio (SDR) view. For all functions in the receiver we described the characteristics that are important while mapping onto reconfigurable hardware. During the mapping of the functions, we have only considered the Montium architecture. In order to perform all receiver processing (excluding the synchronization and control processing part), 3 Montium-tiles are required, when we assume the tiles to run at a clock frequency of 100 MHz. We estimated the processing delay in the receiver of one OFDM symbol to be about 11 μs.

For the Bluetooth communication system we have also analysed the receiver's functionality in the physical layer. The Bluetooth receiver showed to be a fairly simple signal processing part, which can be implemented in a Montium architecture quite well. The computation delays of the different receiver parts seem to be a few clock cycles. The processing of one received bits takes about 130 ns, when a FIR filter with 50 coefficients is applied and

the clock frequency of the MONTIUM-tile is 100 MHz.

5.2 Recommendations and future work

While studying the feasibility of implementing the HiperLAN/2 receiver in the MONTIUM architecture, we discovered some problems. The problems concern mostly square root calculations that are involved in the magnitude calculations of complex values. Moreover the mapping of divisions can be difficult. However, either in software running on a general purpose processor performing divisions should be rather computational complex and intensive. We have to do more research on the functionality needed for divisions and square root calculations. Possible solutions of applying divisions and square root calculations can be the use of lookup tables or the use of general purpose processors. While suggesting implementations of the Bluetooth receiver, we have not considered the channel selection functionality in the physical layer. However the channel selection mechanism is very dynamic, while the subcarrier that contains the information is changing at most every 625 μs, and therefore it would show a perfect example of adaptivity and reconfigurability in reconfigurable hardware. The channel selection mechanism should be studied in detail and implementations should be suggested in order to make the dynamically reconfigurable Bluetooth receiver complete.

Although we have studied some communication systems in a Software Defined Radio perspective, we have not focused on all details.

- First of all, only two different communication standards are subjected to our study. Those two communication standard have been selected, because they are already part of ongoing research at the University of Twente [3]. The output of this research is used to get an idea of all functionality needed in the reconfigurable architecture. However, in future we will also subject a UMTS communication system to our study with typical functions like RAKE-receivers and Turbo-encoders/decoders. Such a communications system is already under research at the University of Twente [1].

- On the other hand, some rough estimations about the number of utilized processing parts are being made. However, the estimations only yield for a specific case with fixed parameters. It would be useful to develop a simulator that can be used to simulate different mapping strategies. This simulator should model the dynamically reconfigurable heterogeneous architecture. The specifications of the heterogeneous architecture like clock frequency, number of processing parts, et cetera, can be controlled by the simulator. In this case one can do better estimations about the number of processing parts needed at a certain clock frequency and the processing delay.

Bibliography

[1] Chameleon project - *Reconfigurable computing in hand-held multimedia computers*, `http://chameleon.ctit.utwente.nl`.

[2] L.F.W. van Hoesel, *Design and implementation of a software defined HiperLAN/2 physical layer model for simulation purposes*, Master of Science Thesis, University of Twente, Enschede, August 2002.

[3] Software-Defined-Radio project - *A Bluetooth-HiperLAN/2 SDR receiver*, `http://www.sas.el.utwente.nl/home/SDR/`.

[4] P.M. Heysters, *The Montium architecture specification*, draft, 25 September 2002.

[5] Bluetooth SIG, *Specification of the Bluetooth System - Core*, Technical Specification Version 1.1, 22 February 2001.

[6] Vincent Arkesteijn, Roel Schiphorst, Fokke Hoeksema, Eric Klumperink, Bram Nauta, Kees Slump, *A Software Defined Radio Testbed for WLAN Front Ends*, PROGRESS workshop, 24 October 2002, Utrecht, the Netherlands.

[7] Lars van Mourik, Roel Schiphorst, Fokke Hoeksema, Kees Slump, *Performance evaluation of a combined Hiperlan/2-Bluetooth digital front-end*, PRORISC workshop, 28-29 November 2002, Veldhoven, the Netherlands.

[8] Roel Schiphorst, Fokke Hoeksema, Kees Slump, *Bluetooth demodulation algorithms and their performance*, 2nd Karlsruhe Workshop on Software Radios, pages 99–106, March 2002.