MASTER

Efficient implementation of a WCDMA RAKE receiver on a programmable platform

Quax, M.M.G.

Award date:
2003

Link to publication
EFFICIENT IMPLEMENTATION OF A WCDMA RAKE RECEIVER ON A PROGRAMMABLE PLATFORM

M.M.G. Quax

Supervisor: prof.dr.ir. J.L. van Meerbergen
Date: June 2003
Efficient implementation of a WCDMA RAKE receiver on a programmable platform

M.M.G. Quax

Design Automation Section
Information and Communications Systems Group
Department of Electrical Engineering
Eindhoven University of Technology

WCDMA; RAKE; programmable digital signal processor; reconfigurable computing; wireless communication; spread spectrum

This thesis is a study of an efficient programmable implementation of a WCDMA Rake receiver. The focus has been on doing a scalable implementation with small distributed memories and low power consumption. The programmable Rake receiver implemented as part of this thesis has been written in AIRT C and functionally simulated and verified using Matlab / Simulink. At VHDL and netlist level the verification is done with Cadence ngsim. The area and power consumption have been estimated with synthesis (Cadence) and power estimation using a wire load model. In this thesis two system design choices are made by mapping the Rake algorithm on an AIRT processor template. The first choice concerns the placement of the alignment memory. By moving the time alignment of the "fingers" (multipath components) from the beginning of a Rake receiver into the maximum ratio combiner, we have shown that not only will we reduce the memory access bandwidth but also the memory area. The second choice influences the exploiting of parallelism. By placing branch conditions not in the control flow but locally in the data path, the algorithm can fully exploit the parallelism in the algorithm. The resulting architecture consists of a mixture of generic functional units, extended functional units (with conditional input) and dedicated functional units for bit level operations.

The results of this thesis are power and area numbers of a programmable Rake receiver:
Area: 0.2 mm$^2$ - 0.12μ CMOS
Power: 6 mW/ 100 MHz/ 0.2 mm$^2$ / 1.2V
Preface

This Master thesis was done at the Eindhoven University of Technology, the Netherlands during September 2002 and May 2003 as part of my M.Sc Electrical Engineering studies at the Department of Electrical Engineering, Information and Communications Systems Group at the Eindhoven University of Technology.

Eindhoven, 17th June 2003
## Contents

1 Introduction  
1.1 Background ............................... 1  
1.2 Thesis objectives .......................... 1  
1.3 Organization of the thesis ............... 1  

2 WCDMA  
2.1 Introduction ................................ 2  
2.2 CDMA basics ............................... 2  
2.2.1 Direct sequence CDMA ...................... 3  
2.2.2 Spreading and despreading ............... 5  
2.2.3 Multiple access .......................... 7  
2.3 Spreading codes ............................ 9  
2.3.1 Properties of PN sequences ............... 9  
2.3.2 M-sequence .............................. 9  
2.3.3 Scrambling codes ......................... 10  
2.3.4 Channelization codes ..................... 11  
2.4 Performance in the presence of interference  
2.4.1 Narrowband interference ................. 12  
2.4.2 Wideband interference .................... 12  
2.4.3 Multiple access interference .......... 13  
2.4.4 Power control ........................... 14  
2.5 Multi path ................................ 15  
2.5.1 Rake receiver ....................... 17  
2.5.2 Channel Estimation ...................... 19  
2.5.3 Maximum Ratio Combining ............. 20  
2.6 Soft and softer hand over ................. 21  
2.7 Synchronization ........................... 23  
2.7.1 Acquisition phase (coarse alignment) .. 23  
2.7.2 Tracking phase (fine alignment) ........ 25  

3 Rake receiver architecture ................. 26  
3.1 Rake? .................................... 26  
3.2 Rake in literature ......................... 26  
3.3 Conventional Rake receiver ............... 28  
3.4 FlexRake receiver ......................... 29  
3.4.1 Stream buffer ........................... 30  
3.4.2 correlator engine ....................... 30  
3.4.3 control .............................. 31  
3.5 Post buffer Rake receiver, ASIC implementation  

©Eindhoven University of Technology 2003  

3.5.1 Chip Aligner .......................................................... 32
3.5.2 PN generator ......................................................... 33
3.5.3 Descrambler .......................................................... 34
3.5.4 Despreader ............................................................ 34
3.5.5 OVSF generator ...................................................... 34
3.5.6 Symbol despreade ................................................... 36
3.5.7 Channel compensator .............................................. 37
3.5.8 MRC ................................................................. 37
3.6 R.E.A.L. DSP implementation ...................................... 38
3.6.1 Input stage .......................................................... 38
3.6.2 Receiver finger ...................................................... 39
3.6.3 Delay block .......................................................... 39
3.6.4 Weighting block .................................................... 40
3.6.5 Combination block ............................................... 40
3.6.6 Rake scheduler ...................................................... 40
3.6.7 Implementation ..................................................... 40
3.6.8 Offset adjustment .................................................. 40
3.6.9 Conclusion .......................................................... 41

4 Designflow .............................................................. 42
4.1 Introduction ........................................................... 42
4.2 Matlab/ Simulink ...................................................... 44
  4.2.1 Associated files .................................................. 44
  4.2.2 S-Function ......................................................... 46
4.3 AIRT Designer ........................................................ 47
  4.3.1 Processor architecture template ................................ 49
  4.3.2 Logic synthesis ................................................... 51
4.4 Behavioral synthesis .................................................. 52
  4.4.1 Behavioral Synthesis Concepts ................................ 52
  4.4.2 Behavioral limitations .......................................... 52
4.5 VHDL / Verilog simulation ......................................... 53
4.6 Power estimations: DIESEL wire load model ....................... 54
  4.6.1 Dissipation in CMOS circuits ................................ 54
  4.6.2 Static power dissipation ....................................... 55
  4.6.3 Dynamic power dissipation ................................... 55
  4.6.4 Calculating power dissipation ................................ 55
  4.6.5 Wire load models ................................................. 57
  4.6.6 Back annotation of wire capacitance ......................... 58
  4.6.7 Back annotation of timing ..................................... 58
  4.6.8 Test bench dependence ......................................... 58
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CDMA classification</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Direct sequence CDMA</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Spreading and despreading process</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>CDMA spectrum</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Spreading operation in DS-CDMA</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>CDMA correlation principle</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>Multiple Access</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>Multiple Access frequency spectrum</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>M-sequence</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>autocorrelation M-sequence</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>Long scrambling code uplink 3GPP</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>Implementation of OVSF code generator</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>Narrowband interference</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>Wideband interference</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>Multiple user frequency spectrum</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>Wideband interference, multiple access</td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>Power control</td>
<td>14</td>
</tr>
<tr>
<td>18</td>
<td>Multi path</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>Multi path, one direct and one reflected path</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>RAKE receiver</td>
<td>17</td>
</tr>
<tr>
<td>21</td>
<td>MRC multipath</td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>Maximum Ratio Combining principle</td>
<td>18</td>
</tr>
<tr>
<td>23</td>
<td>transmitted complex symbol by base station</td>
<td>19</td>
</tr>
<tr>
<td>24</td>
<td>received multipath symbols distorted by the channel</td>
<td>19</td>
</tr>
<tr>
<td>25</td>
<td>non-coherent combining of multipath symbols</td>
<td>19</td>
</tr>
<tr>
<td>26</td>
<td>phase rotated multipath symbols</td>
<td>20</td>
</tr>
<tr>
<td>27</td>
<td>Coherent summation of three channel compensated multipath symbols</td>
<td>20</td>
</tr>
<tr>
<td>28</td>
<td>Softer handover</td>
<td>21</td>
</tr>
<tr>
<td>29</td>
<td>Soft handover</td>
<td>21</td>
</tr>
<tr>
<td>30</td>
<td>Autocorrelation</td>
<td>23</td>
</tr>
<tr>
<td>31</td>
<td>Delay locked loop</td>
<td>25</td>
</tr>
<tr>
<td>32</td>
<td>Garden rake</td>
<td>26</td>
</tr>
<tr>
<td>33</td>
<td>Number of Rake receiver publications</td>
<td>26</td>
</tr>
<tr>
<td>34</td>
<td>Conventional Rake receiver</td>
<td>28</td>
</tr>
<tr>
<td>35</td>
<td>FlexRake implementation of a direct sequence RAKE receiver</td>
<td>29</td>
</tr>
<tr>
<td>36</td>
<td>FlexRake stream buffer</td>
<td>30</td>
</tr>
<tr>
<td>37</td>
<td>Post buffer Rake receiver</td>
<td>32</td>
</tr>
<tr>
<td>38</td>
<td>Chip aligner sub block</td>
<td>33</td>
</tr>
</tbody>
</table>

©Eindhoven University of Technology 2003
Chip aligner ................................................. 33
Time multiplexed pn generator .......................... 34
Descrambler .................................................. 34
Despreader .................................................... 35
OVSF generator ............................................. 35
Symbol despreader ......................................... 36
Channelization code multiplier .......................... 36
Complex multiplier .......................................... 37
VDB memory .................................................. 37
MRC .......................................................... 38
R.E.A.L. implementation of RAKE receiver ............. 38
Timing diagram input stage ................................ 39
Block diagram receiver finger ............................. 39
Designflow .................................................... 42
Mathworks Simulink WCDMA FDD model ............... 44
Simulink sub model of Rake receiver .................... 45
Simulink sub model of Rake receiver .................... 45
Simulink customized interface ............................ 46
ART designer main gui ...................................... 47
ART Designer designflow ................................... 48
ART Designer architectural model ....................... 49
ART Designer architectural model from register to register 50
Timing and functional behavior analysis ................. 51
Cadence NC VHDL ........................................... 53
Power dissipation in a CMOS inverter cell ............... 54
Currents in a basic CMOS circuit for rising and falling output transitions 55
Common memory resource .................................. 60
Distributed memory resources ............................... 60
Memory alignment before despeading, FlexRake implementation ......... 62
Single memory, alignment after despeading .............. 63
Distributed memory, alignment after despeading .......... 64
Memory alignment in Maximum ratio combiner ............ 65
Memory size vs. rake receivers ............................ 67
Memory bandwidth vs. rake receivers ..................... 68
Example of conditional code in Rake algorithm .......... 69
General VLIW architecture ................................ 71
Serial schedule of Rake algorithm on single thread controller 71
MIMD VLIW architecture .................................. 72
Conditional statements in datapath ...................... 73
Conditional schedule ....................................... 74
79 Condition locally in the datapath ........................................ 74
80 Control data in data path of ALU ....................................... 75
81 Time diagram control flow .............................................. 76
82 Serial schedule of MRC operations .................................... 77
83 Parallel schedule of MRC operations .................................. 77
84 Rake processor ............................................................ 79
85 Pie chart area usage ...................................................... 81
List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scalability</td>
<td>59</td>
</tr>
<tr>
<td>2</td>
<td>ALU registers usage</td>
<td>75</td>
</tr>
</tbody>
</table>
Abbreviations

3GPP  3rd Generation partnership project (produces WCDMA standard)
3GPP2  3rd Generation partnership project (produces cdma2000 standard)
AICH  Acquisition indication channel
ART  Algorithm to register transfer
AWGN  Additive white Gaussian noise
BCCH  Broadcast channel (logical channel)
BCH  Broadcast channel (transport channel)
BER  Bit error rate
BPSK  Binary phase shift keying
BS  Base station
BSS  Base station subsystem
BSC  Base station controller
C  Programming language
CA-ICH  Channel assignment indication channel
CCCH  Common control channel (logical channel)
CCH  Common transport channel
CDMA  Code division multiple access
CIR  Carrier to interference ratio
CPCH  Common packet channel
CPICH  Common pilot channel
CRC  Cyclic redundancy check
CS  Circuit switched
CSICH  CPCH status indication channel
CTCH  Common traffic channel
CWTs  China wireless telecommunications standard group
DCA  Dynamic channel allocation
DCCH  Dedicated control channel (logical channel)
DCEF  Dedicated control functional entity
DCH  Dedicated channel (transport channel)
DPCCH  Dedicated physical control channel
DPDCH  Dedicated physical data channel
DS-CDMA  Direct spread code division multiple access
DSCH  Downlink shared channel
DTCH  Dedicated traffic channel
EDGE  Enhanced data rates for GSM evolution
ETSI  European telecommunications standards institute
FACH  Forward access channel
FBI  Feedback information
FDD  Frequency division duplex
FDMA  Frequency division multiple access
FIFO  First in first out
FU  Functional unit
GPRS  General packet radio system
GPS  Global positioning system
GSM  Global system for mobile communications
HLR  Home location register
Interference cancellation
identity
International mobile telephony, 3rd generation networks are referred as IMT-2000 within ITU
Internet protocol
Interference rejection combining
IS-95 evolution standard, (cdma 2000)
US-TDMA, one of the 2nd generation systems, mainly in America
IS-95 cdmaOne, one of the 2nd generation systems, mainly in America and in Korea
Inter symbol interference
International telecommunications union
Medium access control
Multiple access interference
Multicarrier modulation
Minimum mean square error
Mobile station
Mobile services switching center, visitor location register
Minimum phase shift keying
Mobile terminal
Multitone modulation
Multi user detection
Orthogonal frequency division multiplexing
Orthogonal variable spreading factor
Power control
Physical common control channel
Paging channel (logical channel)
Primary common control physical channel
Paging channel (transport channel)
Physical common packet channel
Personal communication system, 2nd generation cellular systems mainly in America, operating partly on IMT-2000
Personal digital cellular, 2nd generation system in Japan
Physical downlink shared channel
Physical layer
Parallel interference cancellation
Paging indicator channel
pseudo random
Physical random access channel
Physical shared channel
Quality of service
Quadrate phase shift keying
Random access channel
Radio access network
Radio frequency
Radio link control
Radio network controller
Radio network sub system
Received signal strength indicator
Resource reservation protocol
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCCP</td>
<td>Signalling connection control part</td>
</tr>
<tr>
<td>SCCPCH</td>
<td>Secondary common control physical channel</td>
</tr>
<tr>
<td>SCH</td>
<td>Synchronization channel</td>
</tr>
<tr>
<td>SDD</td>
<td>Space division duplex</td>
</tr>
<tr>
<td>SF</td>
<td>Spreading factor</td>
</tr>
<tr>
<td>SIC</td>
<td>Successive interference cancellation</td>
</tr>
<tr>
<td>SINR</td>
<td>Signal to noise ratio where noise includes both thermal and interference</td>
</tr>
<tr>
<td>SIR</td>
<td>Signal to interference ratio</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to noise ratio</td>
</tr>
<tr>
<td>STD</td>
<td>Switched transmit diversity</td>
</tr>
<tr>
<td>STTD</td>
<td>Space time transmit diversity</td>
</tr>
<tr>
<td>TCH</td>
<td>Traffic channel</td>
</tr>
<tr>
<td>TD-CDMA</td>
<td>Time division CDMA, combined TDMA and CDMA</td>
</tr>
<tr>
<td>TDD</td>
<td>Time division duplex</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time division multiple access</td>
</tr>
<tr>
<td>TE</td>
<td>Terminal equipment</td>
</tr>
<tr>
<td>TFCI</td>
<td>Transport format combination indicator</td>
</tr>
<tr>
<td>TFI</td>
<td>Transport format indicator</td>
</tr>
<tr>
<td>TPC</td>
<td>Transmission power control</td>
</tr>
<tr>
<td>TS</td>
<td>Technical specification</td>
</tr>
<tr>
<td>TTA</td>
<td>Telecommunications technology association (Korea)</td>
</tr>
<tr>
<td>TTC</td>
<td>Telecommunications technology commission (Japan)</td>
</tr>
<tr>
<td>UE</td>
<td>User equipment</td>
</tr>
<tr>
<td>UL</td>
<td>Uplink</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal mobile telecommunication services</td>
</tr>
<tr>
<td>USCH</td>
<td>Uplink shared channel</td>
</tr>
<tr>
<td>US-TDMA</td>
<td>IS-136, one of the 2nd generation systems mainly in USA</td>
</tr>
<tr>
<td>UTRA</td>
<td>UMTS terrestrial radio access (ETSI)</td>
</tr>
<tr>
<td>UTRAN</td>
<td>UMTS terrestrial radio access network</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very High Speed Integrated Circuit Hardware Description Language</td>
</tr>
<tr>
<td>VLIW</td>
<td>Very low instruction word</td>
</tr>
<tr>
<td>WCDMA</td>
<td>Wideband code division multiple access</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

In the standardization forums, WCDMA technology has emerged as the most widely adopted third generation air interface. Its specifications have been created in 3GPP (the 3rd Generation Partnership Project), which is a joint standardization project of Europa, Japan, Korea, USA and China.

Wideband CDMA (WCDMA) standard is designed to support variable data rate up to 2 Mbits/s to the end user. This puts more demands on the digital signal processing of the receiver and transmitter in the mobile terminals.

In the receiver deals with the more signal processing intensive parts. The receiver has to be able to retrieve as much information, from the mobile channel, as possible, with the constrained that other mobile users limit their transmitted power (reducing multiple access interference). The receiver implemented as part of this thesis is the Rake receiver, which combines coherently different multipaths to one stronger signal.

1.2 Thesis objectives

The main purpose of this master thesis was to design an efficient programmable and scalable processor for WCDMA Rake receiver. The focus has been on doing a scalable implementation with small distributed memories and low power consumption. The programmable Rake receiver has been written in a subset of the programming language C.

1.3 Organization of the thesis

Chapter 1 gives a brief description of the background of WCDMA and the 3G mobile communication standard. Chapter 2 gives a more detailed description of the CDMA and WCDMA algorithms and an introduction to the multipath reception and Rake receiver. Chapter 3 describes different Rake receivers used in WCDMA systems. Chapter describes the designflow, the steps made to come from algorithm to RT level netlist implementation. Chapter 5 describes the architecture choices made and the final processor architecture with the power and area estimations.
2 WCDMA

2.1 Introduction

This chapter illustrates the basic principles of CDMA (Code Division Multiple Access). The scope of the first section is to give a generic understanding of CDMA. Following sections give a more extensive discussion on direct sequence CDMA aspects.

The chapter is divided into eight subsections. Section 2.2 introduces the CDMA concept in general: which criteria the transmitted signal has to fulfill to be divided into the class of spread spectrum signals. The definition and properties of the processing gain is defined. The fundamental properties of CDMA signals, namely multiple access capability, protection against multipath interference, privacy, interference rejection, anti jamming capability, and low probability of interception, are introduced.

In section 2.3 different spreading codes used in CDMA applications are discussed, followed by a more extensive discussion on the performance in the presence of interference (i.e., multipath interference, interference rejection, anti jamming capability and low probability of interception) in section 2.4.

The basic concept of the multipath radio channel and Rake receiver is discussed in section 2.5, followed by presentation of soft and softer handovers. In the last section of this chapter synchronization issues are discussed.

2.2 CDMA basics

Code division multiple access (CDMA) is used in spread spectrum systems to enable multiple access. It is a transmission technique in which the frequency spectrum of a data signal is spread using a code uncorrelated with that user data signal. This code has to be unique to every user. Since the applied codes are selected for their low cross correlation values, it is possible to make a distinction between the different signals at the receiver.

The first usage was in the military field because of the difficulty to jam or detect spread spectrum signals. Nowadays spread spectrum systems are widely used in commercial applications (for instance IS-95, CDMA2000, UMTS).

When the user data signal is multiplied with a code, the bandwidth of the resulting signal is increased. The spectrum is "spread" which justifies the name "spread spectrum". The spreading of the spectrum causes the power spectral density function of the resulting signal to decrease, since the total signal power remains the same. The ratio of transmission and information bandwidth is therefore an important parameter in spread spectrum systems.

The ratio of transmitted bandwidth to information bandwidth is called the processing gain, $G_p$, of the spread spectrum system.

$$\text{BW}_t / \text{BW}_i = G_p \quad (1)$$

Where $\text{BW}_t$ is the transmission bandwidth and $\text{BW}_i$ is the bandwidth of the information data signal. $G_p$, the processing gain which is the same as the spreading factor (SF). It also determines the number of users that can be allowed at the same time in a system.

Different spread spectrum techniques exists: Direct Sequence (DS), Frequency Hopping (FH), Time Hopping (TH) and multi carrier CDMA (MC-CDMA).

A general classification of CDMA is given in Figure 1 on the facing page. There are a number of modulation techniques that generate spread spectrum signals:

- Direct Sequence spread spectrum.

1The bandwidth of the code signal is greater than the bandwidth of the data signal
Figure 1: CDMA classification

The user information bits (symbols) are spread over a wide frequency bandwidth by multiplying the user data bits with a spreading code sequence at a high rate.

- Frequency Hopping spread spectrum.
  The carrier frequency at which the information data signal is transmitted is changed according to the spreading code.

- Time Hopping spread spectrum.
  The information data signal is not transmitted continuously. Instead, the signal is transmitted in short bursts. The time of the bursts are decided by the spreading code.

Hybrid modulation.
When a combination of the two of the above mentioned modulation schemes are combined, then a hybrid modulation is created. The purpose of this is to combine the advantages of the modulation and to combat the disadvantages of the two.

The remaining chapters of the thesis will mainly concentrate on direct sequence (DS-CDMA) and its related subjects.

2.2.1 Direct sequence CDMA

Direct sequence is the most popular spread spectrum technique. The data signal is multiplied with a pseudo random bit sequence, often referred to as pseudo random noise code (PN code) or spreading code.

A PN code is a sequence of bits (see Figure 3 on the next page), valued -1 and 1 (polar) or 0 and 1 (non polar). Such sequences have noise like properties like spectral flatness and low cross correlation values, and thus complicate jamming or detection by non target receivers.

Several families of binary PN codes exist: M sequences\(^2\), Gold codes and Kasami codes. The last two can be created by combining a number of M sequences. See section 2.3.

\(^2\)Maximum length sequence

©Eindhoven University of Technology 2003
An usual way to create a PN code is with shift registers with feed back taps. By putting the feedback taps at specific positions, the output sequence of a shift register is of maximum length. The above mentioned code families have this property. When the length of a shift register is \( n \), the length of the resulting sequence is:

\[
N_{DS} = 2^n - 1
\]  
(2)

In direct sequence systems the length of the code is equal to the spreading factor so:

\[
SF = G_p = N_{DS}
\]  
(3)

This can also be seen in Figure 3, where the spreading process is illustrated. In this example \( N_{DS}=6 \), i.e., each information symbol is spread by a factor of 6. The bandwidth of the data signal is now multiplied with a factor of \( N_{DS} \). The total signal power remains the same, with the result that the power spectral density is lowered.
In the receiver, the received signal is multiplied again with the conjugated (synchronized) version of the spreading code.

Since the spreading consists of -1s and 1s, the despreading operation removes the complete code from the signal and the original data signal is left. Another observation is that the despread operation is the same as the spread operation. The consequence is that a possible jamming-signal in the radio channel will be spread before data-detection is performed, so jamming effects are reduced.

The influences of a possible jamming or interference signal in the radio channel will be spread before data detection is performed. In this way jamming effects are reduced (see section 2.4 and Figure 13 till 16).

In a direct sequence code division multiple access system the modulated information data signal is directly multiplied by a digital, discrete time, discrete valued code signal. Its from this direct multiplication that the direct sequence CDMA gets its name.

\[
BW_{\text{inf}} @ R_s \ll BW_i @ R_c
\]

\[
SF = G_p = \frac{BW_i}{BW_c} = \frac{R_c}{R_s} = \frac{T_b}{T_c} = N_c
\]

2.2.2 Spreading and despreading

Figure 5 shows the basic operations of spreading and despreading for a direct sequence CDMA system. User data is here assumed to be a BPSK-modulated bit sequence of rate \( R_s \), the user data bits assuming the value of ±1. The spreading operation, in this example, is the multiplication of each user data bit with a sequence of 8 code bits, called chips. The resulting spread data is at a rate of \( 8 \cdot R_s \) and has the same random (pseudo-noise like) appearance as the spreading code. In this case the spreading factor is 8. This wideband signal would then be transmitted across a wireless channel to the receiving end.

During despreading the spread user data/chip sequence is multiplied, bit duration by bit duration, with the same 8 code chips used during the spreading of these bits. As shown, the original user bit sequence has been recovered perfectly. Provided (as shown in Figure 6) perfect synchronization between the spread user signal and the (de)spreading code.

The basic operation of the correlation receiver for CDMA is shown in figure 6. The upper half of the figure shows the reception of the desired own signal. The signal is despread with a perfectly synchronize

©Eindhoven University of Technology 2003
code (How to synchronize the code see section 2.7). Then, the correlation receiver integrates (i.e. sums) the resulting products (data · code) for each user bit.

The lower half of figure 6 shows the effect of the despreading operation when applied to the CDMA signal of another user whose signal is assumed to have been spread with a different spreading code. The result of multiplying the interfering signal with the own code and integrating the resulting products leads to interfering signal values lingering around 0.

As can be seen, the amplitude of the own signal increases on average by a factor of 8 relative to that of the user of the other interfering system, i.e. the correlation detection has raised the desired user signal by the spreading factor, here 8, from the interference present in the CDMA system. This effect is termed 'processing gain' and is a fundamental aspect of all CDMA systems, and in general of all spread spectrum systems.

Because of the coding and the resulting enlarged bandwidth, spread spectrum signals have a number of properties that differ from the properties of narrowband signals. The most interesting from communication system point of view are discussed below.

1. Multiple access capability.
   If multiple users transmit a spread spectrum signal at the same time, and at the same spectrum, the receiver will still be able to distinguish between the users provided each user has a unique code that has a sufficiently low cross correlation with the other codes. Correlating the received signal with a code signal from a certain user will then only despread the signal of this user, while the other spread spectrum signals will remain spread over a large bandwidth and behave like random noise. Thus, within the information bandwidth the power of the desired user will be larger than the interfering power provided there are not too many interferers, and the desired signal can be extracted. The multiple access capability is illustrated in figure 7. Multiple users generate a spread spectrum signal from their narrowband data signals. In figure 8 both users transmit their spread spectrum signals at the same time. At the receiver 1 only the signal of user 1 is coherently summed by the user 1 despreader and the user 1 data recovered.

2. Protection against multipath interference.
   In a radio channel there is not just one path between a transmitter and receiver. Due to reflections (and refractions), a signal will be received from a number of different paths. The signals of the different paths are all copies of the same transmitted signal but with different amplitudes, phases, delays, and arrival angles. Adding these signals at the receiver will be constructive at some of the frequencies and destructive at others. In the time domain, this results in a dispersed signal. Spread spectrum modulation can combat this multipath interference; however, the way in which this is achieved depends very much on the type of modulation used.
3. Privacy.
The transmitted signal can only be despread and the data recovered if the code is known to the receiver.

4. Interference rejection.
Cross correlating the code signal with a narrowband signal will spread the power of the narrowband signal thereby reducing the interfering power in the information bandwidth. This is illustrated in figures 13 till 16. The receiver observes spread spectrum signal summed with a narrowband interference. At the receiver the spread spectrum signal is despread while the interference signal is spread making it appear as background noise compared to the despread signal. Demodulation will be successful if the resulting background is of sufficiently weak energy in the despread information bandwidth.

5. Anti jamming capability, especially narrowband jamming.
This is more or less the same as interference rejection except the interference is now willfully inflicted on the system. It is this property, together with the next one, that makes spread spectrum modulation attractive for military applications.

6. Low probability of interception (LPI).
Because of its low power density, the spread spectrum signal is difficult to detect and intercept by a hostile listener.

2.2.3 Multiple access
In CDMA each user is assigned a unique code sequence (spreading code), which it uses to encode its information bearing signal. The receiver, knowing the spreading code sequence, decodes the received signal and recovers the original user data. This is possible because the cross correlation between the transmitted sequence of the desired user and the sequences of the other users are small.

The spectral spreading of the transmitted signal gives to CDMA its multiple access capability.
All users can transmit at the same time, and each is allocating the entire available frequency spectrum for transmission (i.e., uses the same RF bandwidth).
CDMA does not require the bandwidth allocation of FDMA\(^3\), nor the time synchronization of the individual users in TDMA\(^4\). A CDMA user has full time and full bandwidth available, but the quality of the communication decreases with an increasing number of users.

\(^3\)FDMA: Frequency division multiple access
\(^4\)TDMA: Time division multiple access

©Eindhoven University of Technology 2003

Figure 7: Multiple Access
A spread spectrum signal modulation technique must fulfill two criteria:

1. The transmission bandwidth must be larger than the information bandwidth.
2. The resulting radio frequency bandwidth is statistically independent of information signal being sent.

The receiver correlates the received signal with a conjugated synchronized local generated version of the spreading code to recover the original information data signal. This implies that the receiver must know the code that was used to modulate the original data signal.

Correlation of the received baseband spread spectrum signal (rx) with the PN sequence of user 1 only despreads the signal of user 1. The other users produce multiple access noise for user 1.

![Multiple Access frequency spectrum](image)

Figure 8: Multiple Access frequency spectrum

Only that portion of noise produced by the other users falling in the information bandwidth \([-R_s, R_s]\) of the receiver, will cause interference with the desired signal (see section 2.4.2).
2.3 Spreading codes

Spreading codes\(^5\) can be divided into pseudo noise sequences and orthogonal variable spreading codes. PN codes are pseudo random noise sequence codes generated by a shift feedback register. The most common PN codes for DS-CDMA systems can be generated using linear shifted feedback registers.

2.3.1 Properties of PN sequences

A Pseudo-Noise (PN) code sequence acts as a noiselike (but deterministic) carrier used for bandwidth spreading of the signal energy. The selection of a good code is important, because type and length of the code sets bounds on the system capability. The PN code sequence is a Pseudo-Noise or Pseudo-Random sequence of 1's and 0's (or -1's and +1's), but not a real random sequence since it is periodic. The auto-correlation of a PN code must have properties similar to white noise.

Pseudo-Random:

- Not random, but it looks randomly for the user who doesn’t know the code
- Deterministic, periodical signal that is known to both the transmitter and the receiver. The longer the period of the PN spreading code, the more the transmitted signal will look like a truly random binary wave. The more random the signal is the harder it is to detect.
- Statistical properties of sampled white-noise

Length:

- Short code: The same PN sequence for each data symbol \((N_c \cdot T_c = T_s)\)
- Long code: The PN sequence period is much longer than the data symbol period, so that a different pattern is associated with each symbol \((N_c \cdot T_c >> T_s)\)

2.3.2 M-sequence

A logic feedback shift register (LFSR) has all the feedback signals returned to a single input of a shift register (delay line). The LFSR is linear if the feedback function can be expressed as a modulo-2 sum (xor).

\[ f(x_1, x_2, ... , x_n) = c_1 x_1 + c_2 x_2 + ... + c_n x_n \]

Figure 9: M-sequence

The feedback function \(f(x_1, x_2, ... , x_n)\) is a modulo-2 sum of the components \(x_i\) of the shift register cells with \(c_i\) being the feedback connection coefficients (\(c_i=0=\text{open}, c_i=1=\text{connect}\)). An LFSR with \(L\) flip-flops produces sequences that depend upon register length \(L\), feedback tap connections and initial conditions.

---

\(^5\) Sometimes spreading codes are called spreading sequences. In this report, the terms spreading code and spreading sequence are used interchangeably.
When the period (length) of the sequences is exactly $N_c=2L-1$, the PN sequence is called a maximum-length sequences or simply an M-sequence. An M-sequence generated from a linear LFSR has an even number of taps.

If an L-stage LFSR has feedback taps on stages L, k, m and has sequence $a_i, a_{i+1}, a_{i+2}, \ldots$, than the reverse LFSR has feedback taps on L, L-k, L-m and sequence $a_{i+2}, a_{i+1}, a_i, \ldots$.

**Properties**

**Autocorrelation** The autocorrelation function of the M-sequence is $-1$ for all values of the chip phase shift $t$, except for the $[-1, +1]$ chip phase shift area, in which correlation varies linearly from the $-1$ value to $2L-1 = N_c$ (the sequence length).

The autocorrelation peak increases with increasing length $N_c$ of the m sequence and approximates the autocorrelation function of white noise. Other codes can do no better than equal this performance of M-sequence!

- \[ \text{Autocorrelation} \]
- \[ \text{Cross-correlation} \]

**Cross-correlation** Cross-correlation is the measure of agreement between two different codes. Unfortunately, cross-correlation is not so well behaved as autocorrelation. When large numbers of transmitters, using different codes, are to share a common frequency band (multi-users environment), the code sequences must be carefully chosen to avoid interference between users.

2.3.3 Scrambling codes

The autocorrelation properties of the M-sequence cannot be bettered. But a multi-user environment (Code Division Multiple Access) needs a set of codes with the same length and with good cross-correlation properties. Gold code sequences are useful because a large number of codes (with the same length and with controlled crosscorrelation) can be generated, although they require only one "pair" of feed-back tap sets.

Gold codes are product codes achieved by the exclusive or-ing (module-2 adding) of two maximum-length sequences with the same length (factor codes). The code sequences are added chip by chip by synchronous clocking. Because the M-sequences are of the same length as the two base codes which are added together, but are non-maximal (so the autocorrelation function will be worse than that of M-sequences). Every change in phase position between the two generated m-sequences causes a new sequence to be generated.

Any 2-register Gold code generator of length L can generate $2L-1$ sequences (length $2L-1$) plus the two base m-sequences, giving a total $2L+1$ sequences. In addition to their advantage in generating large numbers of...
codes, the Gold codes may be chosen so that over a set of codes available from a given generator the autocorrelation and the cross-correlation between the codes is uniform and bounded.

2.3.4 Channelization codes

The Hadamard-Walsh codes are generated in a set of $N = 2^n$ codes with length $N = 2^n$. The generating algorithm is simple.

$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}$ with $H_0 = [1]$

The rows (or columns) of the matrix $H_N$ are the Hadamard-Walsh codes.

$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$

$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}$

In each case the first row of the matrix consist entirely of 1s and each of the other rows contains $N/2 - 1$'s and $N/2$ 1's. Row $N/2$ starts with $N/2$ 1's and ends with $N/2$ -1's. The distance (number of different elements) between any pair of rows is exactly $N/2$. For $H_8$ the distance between any two rows is 4, so the Hamming distance of the Hadamard code is 4. The bits identifies one row of the matrix (there are $N = 2^n$ possible rows).

All rows are mutually orthogonal:

$$\sum_{k=0}^{N-1} h_{ik} \cdot h_{jk} = 0 \quad \forall i, j$$

(6)

The crosscorrelation between any two Hadamard-Walsh codes of the same matrix is zero, when perfectly synchronized. A possible implementation of an OVSF code generator is shown in figure 12.

In a synchronous CDMA system, e.g. IS-95, this ensures that there is no interferences among signals transmitted by the same station. Only when synchronized, these codes have good orthogonal properties. The codes are periodic, which results in less spreading efficiency and problems with synchronization based on autocorrelation.

©Eindhoven University of Technology 2003
2.4 Performance in the presence of interference

2.4.1 Narrowband interference

The narrowband noise is spread by the multiplication with the PN sequence $p_n$ of the receiver. The power density of the noise is reduced with respect to the despread data signal. Only $1/G_p$ of the original noise power is left in the information baseband ($R_s$). Spreading and despreading enables a bandwidth trade for processing gain against narrow band interfering signals. Narrow-band interference would strongly influence conventional narrowband receivers. The essence behind the interference rejection capability of a spread spectrum system is that the information signal (data) gets multiplied twice by the PN sequence, but the interference signal gets multiplied only once.

![Fig. 13: Narrowband interference](image)

2.4.2 Wideband interference

Multiplication of the received signal with the PN sequence of the receiver gives a selective despread of the data signal (smaller bandwidth, higher power density). The interference signal is uncorrelated with the PN sequence and is spread. This means in the situation of multiple spread spectrum users, each user introduces Gaussian noise. The larger channel bandwidth ($R_c$ instead of $R_s$) increases the received noise power with $G_p$.

![Fig. 14: Wideband interference](image)
2.4.3 Multiple access interference

The detector receives a signal composed of the sum of all users signals, which overlap in time and frequency. Multiple access interference (MAI) refers to the interference between direct sequence users and performance of DS-CDMA systems.

![Multiple user frequency spectrum](image1)

**Figure 15:** Multiple user frequency spectrum

![Wideband interference, multiple access](image2)

**Figure 16:** Wideband interference, multiple access

In a conventional DS-CDMA system, a particular user's signal is detected by correlating the entire received signal with that user's code waveform. The conventional detector does not take into account the existence of MAI. Because of the interference among users, however, a better detection strategy is one of multi-user detection. Information about multiple users is used jointly to better detect each individual user.

Near far problem Each user is a source of interference for other users, and if one is received with more power, than that user generates more interference for the other users (see section 2.4.3). It is important that the receiver (e.g. a basestation) gets the same amount of power from each transmitter. The use of power control ensures that all users arrive at about the same power $P_{RX}$ at the receiver, and therefore no user is unfairly disadvantages relative to others. The signal to noise interference power ratio at the receiver input for $N_u$ simultaneous users is:

$$SNR = \frac{P_{RX}}{(N_u - 1) P_{RX}} = \frac{1}{(N_u - 1)}$$  \hspace{1cm} (7)
2.4.4 Power control

Tight and fast power control is perhaps the most important aspect in WCDMA, in particular on the uplink. Without it, a single overpowered mobile could block a whole cell. Figure ?? depicts the problem and the solution in the form of closed-loop transmission power control. Mobile stations MS1 and MS2 operate within the same frequency, separable at the base station only by their respective spreading codes. It may happen that MS2 at the cell edge suffers a path loss, say 70 dB above that of MS1 which is near the base station BS. If there were no mechanism for MS1 and MS2 to be power-controlled to the same level at the base station, MS1 could easily overshoot MS2 and thus block a large part of the cell, giving rise to the so-called near-far problem of CDMA. The optimum strategy in the sense of maximizing capacity is to equalize the received power per bit of all mobile stations at all times, at the receiving base station.

While one can conceive open-loop power control mechanism that attempt to make a rough estimate of path loss by means of a downlink beacon signal, such a method would be far too inaccurate. The prime reason for this is that the fast fading is essentially uncorrelated between uplink and downlink, due to the large frequency separation of uplink and downlink band of the WCDMA FDD mode. Open-loop power control is, however, used in WCDMA but only to provide a coarse initial power setting of the mobile station at the beginning of a connection.

The solution to power control in WCDMA is fast closed-loop power control, also shown in figure 17. In closed-loop power control in the uplink, the base station performs frequent estimates of the received Signal-to-Interference Ratio (SIR) and compares it to a target SIR. If the measured SIR is higher than the target SIR, the base station will command the mobile station to lower the power; if it is too low it will command the mobile station to increase its power. This measure-command-react cycle is executed at a rate of 1500 times per second for each mobile station and thus operates faster than any significant change of path loss could possibly happen and, indeed, even faster than the speed of fast Rayleigh fading for low to moderate mobile speeds. Thus closed-loop power control will prevent any power imbalance among all the uplink signals received at the base station.

The same closed-loop power control technique is used in the downlink, though here the motivation is different: on the downlink there is no near-far problem due to the one-to-many scenario. All the signals within one cell originate from the one base station to all mobiles. It is, however, desirable to provide a marginal amount of additional power to mobile stations at the cell edge, as they suffer from increased other-cell interference.

Figure 17: Power control
2.5 Multi path

Radio propagation in the mobile channel is characterized by multiple reflections, diffractions and attenuation of signal energy. In these environments there are more than one paths from the transmitter to the receiver as shown in figure 18. Such multipaths may due to:

- refractions
- buildings or other objects

Figure 18: Multi path

Multi paths may result in fluctuations in the received signal level (fading). Each path has its own attenuation and time delay profile. Assume that the receiver is synchronized to the time delay and RF phase of the direct path.

Suppose two discrete paths: a direct path and only one non direct path (delayed by a time $\tau$ compared to the direct path).

Figure 19: Multi path, one direct and one reflected path

The signal at the receiver can be expressed as:

$$RX(t) = RX_{direct} + RX_{reflect} + N$$

$$= A \cdot d_i(t) \cdot p(t) \cdot \cos(\omega_0 t) + \alpha \cdot A \cdot d_i(t - \tau) \cdot p(t - \tau) \cdot \cos(\omega_0 t + \theta) + n(t)$$

(8)

For the receiver, synchronized to the direct path signal, the output of the correlator, can be written as:

$$d_r(t) = N_e T_e = \int_0^{N_e T_e} p(t) \cdot RX(t) dt$$

(9)
The PN sequence has an autocorrelation function with the property (see section 2.3):

- \( \text{PN}(t) \cdot \text{PN}(t) = 1 \)
- \( \text{PN}(t) \cdot \text{PN}(t-T) \neq 1 \)

The signals that arrive from the non direct channel are not synchronized to the local generated PN code, of the direct path, and is thus rejected. Multipath signals that are delayed by a chip period or longer relative to the desired signal (reflections) are essentially uncorrelated and do not contribute to multipath fading. The spread spectrum effectively rejects the multipath interference (see sections 2.4.2 and 2.4.3).

\[
d_s(t = N_c T_c) = d_t + n_0
\]

with \( n_0 \) = noise and multipath interference.
2.5.1 Rake receiver

A direct sequence spread spectrum signal waveform is well matched to the multipath channel. In a multipath channel, the original transmitted signal reflects from obstacles such as buildings and mountains (see figure 18). The receiver receives not only the original transmitted data signal but also a number of delayed and attenuated versions of the same transmitted data signal. If the signals arrive more than one chip apart from each other, the receiver can resolve them. This means that, from a multipath signal’s point of view, other multipath signals can be regarded as interference and they are suppressed by the processing gain, i.e. the cross correlation between the original data signal and a delayed version of the original signal is small.

\[
\text{average}(p n_r(t) \cdot p n_r(t + \tau)) \ll 1 \quad [\forall t]
\]

A further benefit is obtained when the resolved multipath signals are coherently combined using a Rake receiver. The resulting signal has a larger signal to interference ratio.

A Rake receiver can be seen as a set of correlators, each receiving one (delayed) multipath signal. After descrambling and despreading, by the code generators and correlators, the signals are coherently combined using, for example, Maximum Ratio Combining. The received multipath signals are fading independently, therefore the diversity order increases and the performance is improved.

Figure 20: RAKE receiver

Figure 20 illustrates the principle of a Rake receiver. After spreading and modulation the signal is transmitted, and passes through a multipath channel, which can be modelled by a tapped delay line (i.e., the reflected signals are attenuated and delayed in the channel).

Figure 21: MRC multipath

In figure 21 three multipath components are shown. These different propagation paths have each different
delays ($\tau_1$, $\tau_2$, $\tau_3$) and gain (attenuation) factors ($a_1$, $a_2$, $a_3$).

The Rake receiver has a receiver finger for each different propagation path. In each finger, the received signal is correlated by a spreading code, which is time aligned with the delay of the multipath signal. After despreading the signals are weighted and time aligned combined.

Figure 22: Maximum Ratio Combining principle

In figure 22 the principle of maximum ratio combining is presented. That is, each signal is weighted by the path gain/attenuation and combined. Due to the mobile environment, i.e. scattering, the channel transfer function will change, the delays and attenuation factors will change as well. Therefore, it is necessary to measure the time delay line profile and reallocate Rake fingers whenever the delays have changed by a significant amount. Small scale changes, less than one chip, are compensated by a code tracking loop, which tracks the time delay of each multipath component (see section 2.7.2).
2.5.2 Channel Estimation

The Channel Estimation keeps track of the phase and amplitude changes of the different active multipath components by despeading the pilot channel (CPICH in downlink of 3GPP).

Figure 23: transmitted complex symbol by base station

Figure 24 shows how the transmitted symbol in figure 23 are distorted in phase and amplitude by the channel.

Figure 24: received multipath symbols distorted by the channel

The resulting symbol in case of non-coherent combining is presented in figure 25.

Figure 25: non-coherent combining of multipath symbols

By calculating the phase distortion offset for each multipath, the received symbols can be phase aligned (see figure 26) and coherently combined (see figure 27).
2.5.3 Maximum Ratio Combining

The Maximum Ratio Combining combines the phase rotated symbols into one stronger symbol.

Figure 26: phase rotated multipath symbols

Figure 27: Coherent summation of three channel compensated multipath symbols
2.6 Soft and softer handover

When a mobile station is in the overlapping cell coverage area of two adjacent sectors of a base station, there has to be a handover action. The mobile station communicates with the base station via two separate air interface channels, one for each sector. The mobile stations requires to receive both signals, and correlates signals with two corresponding separate codes, thus the mobile station can distinguish the two different signals.

![Figure 28: Softer handover](image)

In the uplink direction a similar process takes place at the base station: The code channel of the mobile stations is received in both sectors, then routed to the same baseband Rake receiver and the maximum ratio combined, in the usual way. During softer handover only one power control loop is active per connection. Figure 28 shows the softer handover scenario.

![Figure 29: Soft handover](image)

During soft handover, a mobile station is in the overlapping cell coverage of two sectors belonging to different base stations. As in softer handover, the communications between mobile station and base station takes place concurrently via two air interface channels, each base stations has one separate channel. As in softer handover, both channels (signals) are received at the mobile station by a Rake receiver and maximum ratio combined. From the mobile station point of view, there are not many differences between soft and softer handover.

In the uplink direction, however, soft handover differs significantly from softer handover, the code channel of the mobile station is received from both base stations, but the received data is then routed to the radio network controller (RNC) for combining. During soft handover two power control loops are active per connection, one for each base station.

The mobile stations and base stations need to have additional resources to handle soft handover actions.

- Additional Rake receiver in the base stations
- Additional transmission links between base stations and radio network controller (RNC)
- Additional Rake fingers in the mobile stations

---

6Radio Network Controllers (RNC) are equipment that interface with the core network, control the radio transmitters and receivers in a node, and perform other radio access and link maintenance functions (such as soft handoff) in a 3G wireless network.
Soft and softer handovers can also take place in combination with each other. These specific CDMA handover types are needed for similar reasons as closed loop power control. Without soft/softer handover there would be a near far problem, a mobile station penetrating from one cell deeply into an adjacent cell without being under power control of the adjacent cell.
2.7 Synchronization

If the received signal is multiplied by a despreading code sequence \( p_n^r \), different from the one used in the modulator \( p_n \), the multiplier output becomes:

\[
d_t = RX_b \cdot p_n^r = (d_t \cdot p_n^t) \cdot p_n^r
\]  
(13)

In the receiver, detection of the desired signal is achieved by correlation against a local reference spreading code sequence \( (p_n^t) \). For such communications in a multi user environment, the transmitted data \( d_t \) may not be recovered by a user, that doesn’t know the spreading code \( (p_n^t) \) used at the transmitter. Therefore:

\[
\text{Crosscorrelation} R_c(t) = \text{average}(p_n^t \cdot p_n^r) << 1 \quad \forall t
\]  
(14)

The cross correlation between \( p_n \) and other code sequence has to be small.

For its proper operation, a spread spectrum communication system requires that the locally generated PN sequence \( (p_n^t) \), is synchronized to the PN sequence generator of the transmitter \( (p_n^t) \), both in time as rate. Due to the sharp peak in the autocorrelation function, a misalignment in the PN sequence of \( \frac{T}{2} \) gives a loss of a factor 2 in processing gain.

\[\text{Figure 30: Autocorrelation}\]

The process of synchronizing the locally generated conjugated PN sequence with the received PN sequence is usually accomplished in two stages:

- Acquisition
  This stage consists of bringing the two spreading signals into coarse alignment with one another.

- Tracking
  Once the received PN sequence has been acquired, this stage takes over and continuously maintains the best possible waveform fine alignment by means of a feedback loop. This essential to achieve the highest correlation power and thus the highest processing gain at the receiver.

2.7.1 Acquisition phase (coarse alignment)

The acquisition problem is one of searching throughout a region of time and frequency (chip, carrier) in order to synchronize the received spread spectrum signal with the locally generated PN sequence \( (p_n^t) \). Since the despreading process typically takes place before carrier synchronization, and therefore the carrier is unknown at this point, most acquisition schemes utilize non coherent detection.
A common feature of all acquisition methods is that the received signal and the locally generated PN sequence are first correlated with a coarse time step (mostly $T_{e_1}$) to produce a measure of similarity between the two. This measure is then compared to a threshold to decide whether the two signals are in synchronism. If the signals are synchronized, a verification algorithm is started. To prevent false locking, the acquisition algorithm has to dwell for some time to test the synchronism. After verification the tracking loop takes over.

For proper synchronization, a peaked autocorrelation is required from the PN sequence.

**Matched filter (parallel)** A matched filter calculates the correlation function at each sample timestep ($\frac{1}{2}T_c$). This implementation of a matched filter is the fastest, i.e. short acquisition time, but also requires the most hardware. The hardware increases with the length of the PN sequence and oversampling ratio $s$.

**Sliding/active correlator (serial)** The sliding/active correlator is based on the correlation result of one active correlator. The correlator cycles through the time uncertainty, usually in discrete time intervals of $T_{e_2}$ seconds or less. An active correlator implementation of a matched filter, needs an integration period of $T_{s} = N_e \cdot T_e$ to calculate one point of the correlation function. After each integration interval the correlator output is compared with a threshold value to determine if the known PN sequence is present. If the threshold is not exceeded then the locally generated PN sequence is advanced by $T_e/2$ seconds and the correlation process is repeated.

The benefit is that this implementation uses less hardware as the complete parallel implementation. But the acquisition time increases with a factor $N_e$ (length of code sequence). This becomes unacceptable long for long codes (large $N_c$).

**Serial/parallel synchronization** This implementation uses more correlators in parallel with PN sequences spaced half chip, $\frac{1}{2}T_c$, apart. After integration period of $N_e \cdot T_e$ the results of the correlator outputs are compared. The correlation function is calculated for three successive points in time (spaced one half chip apart). When none of the outputs exceeds the threshold, the locally generated PN sequences are advanced over $3\cdot T_e$ seconds. When one or more outputs exceed the threshold, the largest correlation value is selected. For a search implementation with three correlators in parallel, the acquisition time is reduced by the same factor three. The search time is reduced at the expense of a more complex and costly implementation.
2.7.2 Tracking phase (fine alignment)

The tracking algorithm is instantiated after the acquisition phase, it makes sure that the locally generated PN sequence is in line with the received data signal. If the locally generated PN sequence is not optimal in synchronism with the received data signal, the PN generator has to be adjusted. This is needed to maximize the processing gain.

**Delay Locked Loop (DLL)** The locally generated PN code sequence $p_{nr}(t)$ of the tracking loop has an offset in phase from the incoming $pn(t)$ by a time $\tau$, with $|\tau| < T_e/2$. In the DLL, two PN sequences $p_{nr}(t + \frac{T_e}{2} + \tau)$ and $p_{nr}(t - \frac{T_e}{2} + \tau)$ are delayed from each other by one chip time ($T_e$). The early and Late outputs are the evaluation of the crosscorrelation function at two timepoints:

$C_{early} = R_a(\tau - \frac{T_e}{2})$

$C_{late} = R_a(\tau + \frac{T_e}{2})$

When $\tau$ is positive, the feedback signal $Y(t)$ instructs the chip aligner to select not the chip(i) but chip(i+1) as the next incoming chip. The same holds when the feedback signal $Y(t)$ is negative, the chip aligner has to select chip(i-1).

![Diagram of Delay Locked Loop](image)

Figure 31: Delay locked loop
3 Rake receiver architecture

3.1 Rake?

The Rake receiver got its name from its inventors R. Price and P. Green in 1958. When a signal is received over a multi-path channel, the multi-path components appear at the Rake receiver as multiple branches. These multipaths are coherently combined in the Rake receiver, this model can be abstracted as an ordinary garden rake see figure 32. It is from this figure the Rake receiver got its name.

![Figure 32: Garden rake](image)

3.2 Rake in literature

Since the first publication of Rake receiving in 1958, the amount of publications is increased exponential (see figure 33). The number of publications increased since it was possible to make efficient CDMA implementations for personal communications (IS-95).

![Figure 33: Number of Rake receiver publications](image)
In literature we find different approaches for Rake receiver architectures. The approaches can be divided in the way that the solutions reuse or share hardware between fingers, i.e. space-time computation considerations.

- The presented conventional Rake receiver is a non optimized approach of the implementation of the Rake algorithm.

- DSP implementation is a direct implementation of the conventional Rake algorithm on a signal processor. Because of the huge amount of bit level processing (code generation, correlation) the DSP solutions are not optimal. On the other hand the algorithm, which is a very control intensive (conditional statements), is easy to implement in software.

- The FlexRake receiver, by Digital and Computer Systems Laboratory from the Tampere University of Technology, is an ASIC solution that reuses the hardware for code generation and correlation. It also moves the alignment of the symbols, for coherent combining, to the front. It makes use of a single memory to store the incoming chips.

- Post Buffer approach, is based on the principle of sharing the code generators and correlators, same as the FlexRake. But the Post Buffer approach does the alignment for coherent combining after the correlation in the maximum ratio combiner. In this way reducing the memory access bandwidth.
3.3 Conventional Rake receiver

In a conventional Rake receiver each multi-path component ("finger") has a dedicated Rake finger device consisting of scramble/channelization code generator, correlator, integrator and FIFO (see figure 34). The separate code generators and integrators are time aligned with their respective multi-path component. The code generators are configured with the code offset of their respective multipath. With the increase in the number of multi path components that have to be combined, more dedicated Rake finger are needed. The symbols are stored in a FIFO memory to be time aligned before they are sent to the Maximum Ratio Combiner (MRC). Each Rake finger device operates at an oversampling multiple of chip rate (3.84 MHz in WCDMA).

A typical architecture of a direct sequence code division multiple access (DS-CDMA) receiver is shown in figure 34:

![Figure 34: Conventional Rake receiver](image)
3.4 FlexRake receiver

L. Harju, M. Kuulusa and J. Nurmi have designed a "Flexible Rake receiver architecture for WCDMA Mobile Terminals" called FlexRake[XXX]. The FlexRake receiver is much more efficient than the standard conventional Rake receiver approach. The main optimization is that FlexRake uses one big circular sample buffer to store the incoming I/Q samples and only one correlator engine that works time-multiplexed between the Rake fingers that are active. A detailed block diagram of the FlexRake receiver is shown in the figure below.

![FlexRake diagram](image)

**Figure 35: FlexRake implementation of a direct sequence RAKE receiver**

The FlexRake consists of a sample buffer and a correlator engine. The correlator engine works time multiplexed so that it correlates one sample (chip) for each active finger in turn. This means that to be able to process up to eight active fingers, the correlator needs to be eight times faster than the conventional Rake receiver that has one correlator for each active finger. The correlator engine consists of code generators (for both scrambling and channelization code), complex multiplier, a number of integration registers (one for each active finger) and a FIFO buffer for the correlated symbols (see figure 3.4).

The correlator performs a complex multiplication of the I/Q samples with the combined scramble and channelization code. The result is accumulated with the previous correlated sample and stored in the integration registers for that particular finger. Then the next finger is processed. When one finger has been integrated SF-times (where SF is the spreading factor) the resulting symbol is stored in the symbol FIFO to later be combined with the corresponding symbol from the other active fingers.

The sample buffer consists of a data buffer and two address generators. The data stored in the buffer are the I/Q samples coming in from the ADC (via a pulse shaping filter). The sample buffer is implemented as a time sliding window with three parts (see figure 36): write window, pre-window and post window. The write window prevents the pre-window from being overwritten. The pre-window makes it possible to catch a new multi path finger that has much shorter delay than the current active fingers. The post window is big enough so that it can contain I/Q samples for different fingers that have the longest supported delay between them (in WCDMA this is 77 μs).
3.4.1 Stream buffer

The Stream buffer contains a sample buffer and two address generators. The sample buffer stores the I/Q sample pairs coming from the pulse shaping filtering. The sample buffer, depicted in figure 36, can be comprehended as a time sliding window that is divided into three parts: write window, pre window and post window. The write window allows writing to the buffer without overlapping the pre window and the post window needed to carry out multipath read accesses. These read and write accesses are interleaved in time in order to avoid the need of concurrent memory accesses. Whereas the post window contains the I/Q samples within the longest supported delay spread, the purpose of the pre window is to add headroom for the movement of the first arriving multipath components. Even if the delays shorten considerably, the pre window ensures that multipath samples are not lost because they can be despread from the pre window.

![Figure 36: FlexRake stream buffer](image)

Circular address generator provides a stream of sequential sample buffer cursor and write addresses. The cursor address points to the beginning of the post window and it is incremented periodically after each processing cycle that is equal to the chip duration. Similarly, the write address generator is employed for fetching I/Q multipath samples for correlation from the sample buffer. It contains a number of offset address registers which are controlled by multipath delay estimates. The number of offset address registers \( L \) corresponds to the maximum number of tracked multipath components. As shown in figure 36, the effective read addresses are calculated by summing two values: the cursor address and an offset value.

Each multipath component is read to the correlator engine one at a time for despooling. After each processing cycle the sample buffer cursor and write addresses are incremented, the offset values can be updated, and new I/Q samples are written to the sample buffer. A processing cycle in the stream buffer contains a number of read and write accesses that correspond to the number of the tracked multipath components and the oversampling ratio, respectively.

3.4.2 Correlator engine

Correlator engine contains a complex correlator, code generators for channelization and scrambling codes, a number of integration registers, and a FIFO buffer for symbol dumps. The number of integration registers \( N_{\text{reg}} = L \) and for three parallel code channels \( N_{\text{ireg}} = 3L \).

The correlator performs a complex valued correlation of the I/Q multipath samples with a combined OVSF/Gold code produced by the two code generators. Partial symbol integration results of each multipath components are stored in an integration register. Since the I/Q multipath samples are read from the same buffer sequentially, all correlations can be performed using the same code phase. When multicode transmission is employed, L integration registers and a dedicated channelization code is assigned for each additional code channel. After correlating over one symbol period the final symbol dumps are stored in the FIFO buffer. It
is important to note that since the multipath components are despread sequentially, L symbol dumps for a transmitted data symbol appear in a certain sequential order.

One processing cycle in the correlation engine is divided into a number of correlation cycles. On each correlation cycle, one (single code) or multiple (multi code) correlations are performed with each I/Q multipath sample. Thus for four multipath components (L=4) and three parallel code channels (N_{code}=3), one processing cycle in the correlation engine may include up to 12 correlation cycles.

### 3.4.3 control

Typically, new offsets are generated every 10 ms, i.e. on a time frame basis. It is assumed that the offset update rate is sufficiently fast to avoid the need for delay lock loop (DLL) for code tracking.

Furthermore, the control unit has three operational modes: receiver initialization, steady state reception and sleep modes.

There are a number of advantages gained from the FlexRake receiver architecture. First is the high flexibility of the multipath allocation because multipath components are tracked simply with the offset values and by allocating a dedicated integration register.

Furthermore, the sample buffer pre-window allows the tracked multipath to move to earlier positions in the delay spread without being lost, i.e. negative offset values can be used. This ensures that no I/Q multipath samples are lost even if the delay profile is rapidly changing.

Another advantage is that the OVSF/Gold code generators need not be time aligned separately according to the multipath delays.

Furthermore, multicode reception is more straightforward because the same I/Q multipath sample can be correlated with multiple spreading codes and thus it is not necessary to perform several reads from the same buffer address. Since the operation of the FlexRake receiver is symbol synchronous, the symbol dumps of each multipath component are completed sequentially in time. This facilitates the implementation of the further processing operations, such as channel estimation and channel correction/combining.
3.5 Post buffer Rake receiver, ASIC implementation

The Post buffer Rake receiver takes a different approach to the buffering problem. If the oversampling factor \(O_s\) is eight (8) and the maximum delay \(D_e\) between two fingers is 296 \(T_e\), then the VDB buffer used in the FlexRake receiver needs to be \(VDB_{size} = O_s \cdot D_e = 2368\) I/Q samples large. With a sample width of 2·6 bits (I+Q), the total memory size is 28416 bits. The access rate to/from the pre-VDB memory will be \((O_s+N_{fingers})/T_e=61.44\) MHz, where \(N_{fingers}\) is the number of active fingers.

If we instead process each finger individually, and keep track of their respective delay \((\tau)\), then we can do the time alignment at the very end instead. Since the time alignment is done at symbol rate instead of sample rate, the data access rate to/from the post VDB memory is much lower (4 to 256 times lower depending on the current spreading factor). The memory size used by the post buffer Rake receiver is \(t_{max}/SF_{min} \cdot 2\cdot Symbol\_width = 4736\) bits.

To be able to handle each finger individually in a time multiplexed manner, each block needs to know each fingers respective delay \((\tau_{finger})\). Enclosed with each chip and symbol is their delay value. This eliminates the need for each block to store the different delay values for each finger.

![Figure 37: Post buffer Rake receiver](image-url)

The Post buffering Rake receiver is designed to handle 8 fingers with a maximum delay of 296 \(T_e\) and 6 different scrambling codes (to be able to receive and combine information from six different base stations at once). It is also easy to include more than one dedicated physical channel by instantiating more than one Despreader. In figure 37 the overall structure of the Post Buffering Rake receiver is shown.

3.5.1 Chip Aligner

The I/Q samples coming from the ADC and pulse shaping filter are oversampled by a factor 8 compared to the chip rate. This is so to make it possible for the multi path tracker to get high resolution of the finger delays \((\tau)\). Each finger has a \(\tau\) register that stores the multi path delay value received from the Tracker. This register also stores information about whether the finger is active or not. A comparator compares the three bit circular counter with the three least significant bits of the \(\tau\) register, if equal the prompt and early/late I/Q sample is stored in the prompt respectively the early register, the late register copies the old early register I/Q sample (see figure 38). The early, prompt and late I/Q samples are used by the Tracker. The prompt I/Q sample is also used as the output I/Q chip used by the descrambler.

The Chip aligner block consists of 8 separate sub blocks, one for every finger, that have the same inputs and have the outputs connected together via a demultiplexer controlled by the counter. The early/late samples are taken directly from the input I/Q samples, and the prompt samples are delayed for four clock cycles via...
shift registers. The Frame-sync is also delayed by four cycles (see figure 39). The counter is synchronized to the Frame-sync.

![Diagram](image1)

**Figure 39: Chip aligner**

### 3.5.2 PN generator

The downlink scrambling code (PN) generator block is implemented to generate up to eight separately phase shifted scrambling sequences by time multiplexing. The X and Y register for each separate finger is stored in a register file, where each position is initialized when frame sync for that specific finger occurs. The X register file is initialized according to the scrambling code phase used by the transmitting base station(s). The Y register file entry is initialized to all ones.

The main part of the PN generator block are two shift registers and number of XOR blocks (see section 2.3 for a detailed description of the code generation). The resulting I/Q code value is rotated 45 degrees to make the implementation of the Descrambler easier.
3.5.3 Descrambler

The despreader correlates the incoming I/Q chip with the scrambling code coming from the PN generator. Since the scrambling code has been rotated so that \( S_{dl} \in \{1, 0\}, (-1, 0), (0, 1), (0, -1) \) the despreader can be heavily optimized to only handle multiplication with \((\pm 1, 0)\) and \((0, \pm 1)\). This is easily done with multiplexers and negators (see figure 41).

3.5.4 Despreader

The Despreader is a structural block that connects OVSF-generator, Symbol Despreader, Channel Compensator, MRC and VDB memory (see figure 42). The input to the Despreader is the descrambled chip data (for each active finger), and the outputs are combined symbols.

The Despreader block is instantiated more than once to handle more than one dedicated physical channel.

3.5.5 OVSF generator

The recursive OVSF generator described in section 2.3.4 can due to some symmetries, easily be implemented with combinational logic based on (Eq. 15). All signals in figure 43 are one bit wide.
Figure 42: Despreader

\[ Seq_{f,n} = n - \tau_f \]
\[ Ch_{code,f,n} = 2 \cdot \sum_{i=0}^{7} Seq_{f,n}(i) \cdot \text{Code}_f(\log_2(SF) - 1 - i) - 1 \]  \hspace{1cm} (15)

The synthesis tool can optimize this to less than 90 NAND2 equivalent gates. This is much less than if one should use a software generated lookup table to store the code sequence.

Figure 43: OVSF generator
3.5.6 Symbol despreader

The Symbol despreader despreads the incoming I/Q chips at chip rate to I/Q symbols at symbol rate $R_s$ (chip rate/SF). The chips are first multiplied with the channelization code ($Ch_{code}$), then accumulated over SF number of chips. The partial accumulated symbols from each finger are stored in their respective position in the accumulation register file until SF number of chips from the current finger have been accumulated, the symbol is then transferred to the symbol register file. The symbol register file acts as a symbol alignment buffer that sends out all completely accumulated symbols from the currently active fingers in bursts, starting with finger 0. The outgoing $\tau_f$ is scaled down according to the current channel spreading factor (SF).

![Symbol despreaders](image)

**Figure 44: Symbol despreaders**

The symbol despreaders handle up to eight different fingers time multiplexed to reduce the hardware cost. The channelization code correlator (complex multiplier) is implemented with negators and multiplexors as shown in figure 45.

![Channelization code multiplier](image)

**Figure 45: Channelization code multiplier**
3.5.7 Channel compensator

The Channel compensator multiplies the Channel Estimate calculated by the Channel Estimator (see section 2.5.2) with the accumulated symbols.

![Complex multiplier diagram](image)

Figure 46: Complex multiplier

3.5.8 MRC

The Maximum Ratio Combiner (MRC) block combines the symbol from each active finger into a single stronger symbol (see figure 27). Since the fingers may be separated in time with up to $296\cdot T_c = 77\mu s$ we need to buffer the partial accumulated symbol until all active fingers have arrived. The maximum delay between the same symbol from different fingers is $T_{\text{symbol}} = 296\cdot T_{\text{chip}}/\text{SF}_{\text{min}} = 292\mu s$ this results in the maximum number of delayed symbols being 74 (for SF=4). The MRC block stores the partial accumulated symbols in a Variable Delay Buffer (VDB), which is implemented in a similar way to the VDB in the FlexRake (see figure 47).

![VDB memory diagram](image)

Figure 47: VDB memory

When the MRC receives a symbol, it first reads the partially accumulated value from memory, it then accumulates the received symbol with the stored value, and then stores the new accumulated sum at the same memory position. If the received symbol was from the last of the active fingers, the accumulated value is sent to the next block, and the memory position is cleared. The read and write address is generated by subtracting the delay value $(T_{\text{symbol}}, f)$ from a circular generated address counter.
3.6 R.E.A.L. DSP implementation

The Rake receiver is based on diversity combining approach of multipath reception. The structure of the Rake receiver, which is used in the DSP implementation is shown in figure 49.

![Figure 49: R.E.A.L. implementation of RAKE receiver](image)

It has several receiver fingers. Each finger is tuned to receive signal from a specific path and then all received signals are combined. The main parts of the Rake receiver consists of:

- Input stage
- Receiver finger
- Delay block
- Weighting block
- Combination block
- Rake scheduler

These blocks will be discussed in the next sections.

3.6.1 Input stage

The input of the Rake receives oversampled chips coming from an ADC, each chip is oversampled by the oversampling factor used in the ADC. These samples are first processed by the Input stage. This stage selects one sample for each incoming chip. This selection is made using control information, this information is supplied by the tracking module. The selected sample is then supplied to a Rake finger.
3.6.2 Receiver finger

The received signal is first descramble and then despread the signal from the multi path it is tuned. The Receiver finger consists of separate tasks:

- Generation and synchronization of the local scrambling and channelization codes
- Multiplication of the input signals with these local generated codes and to demodulate the input signals
- Integration of the resulting samples for one symbol period and dumping the integration value at the end of the symbol time

3.6.3 Delay block

Each finger demodulates a different multipath, with a different delay offset. The Delay block is needed in order to coherently combine the symbol values from each Rake finger. Each symbol value is delayed by an amount determined by the path delay to which that finger is tuned.
3.6.4 Weighting block

Each multipath suffers from phase change, amplitude attenuation depending on the characteristic of the path. The weighting block provides correction for the phase distortion. At the same time the Weighting block provides signals with a larger signal power a larger gain in amplitude. In this way the strongest signal gets maximum gain and weaker signals get a lower gain. This is done to get maximum ratio combining.

3.6.5 Combination block

After the weighting operation, the combination block sums the signal coming from all fingers in a coherent manner. After combination, serialization is performed on the I/Q to obtain the data stream.

3.6.6 Rake scheduler

The Rake scheduler is a control block specific to each Rake receiver. This acts as an interface between the Rake receiver and other parts of the receiver. Another task of this block is to control the internal control signals of the Rake receiver.

3.6.7 Implementation

This section discusses the details of the DSP implementation on a REAL dsp. The Rake finger is the first part selected to be implemented on the DSP, because it is the most computationally expensive part of the receiver.

3.6.8 Offset adjustment

If the synchronization tracking module detects a change in path delay it sends a signal to the corresponding finger which tells whether code generators are leading or lagging. The code generators must be capable of shifting their current status backward or forward by one bit. Before generation of 8 new bits of scrambling code, the program checks for an offset. If it is present, an offset routine is called. This routine simply calculates the new/old bit for each shift register and shifts the registers forward/backward by one bit depending on the offset direction. In this way offset adjustment for the scrambling code is achieved.

Since the offset signal is updated once in a time slot period (2560 chips) and the maximum symbol period possible is 256 chips, we will never require more than 1 shift in a symbol period. Therefore the limitation of the approach, that it can accommodate only 8 shifts in a symbol period, will not have any adverse effect.

Another important point here is that we check for the offset only once in the loop (in the beginning).
3.6.9 Conclusion

One complete receiver finger on a R.E.A.L. DSP and for this finger the load is around 100 Mcycles/sec (@ chiprate = 4 Mchips/sec). A 20 percent reduction in cycle count was achieved after optimizations. The total number of fingers in a mobile receiver will depend upon the number of downlink channels used and also the number of paths to be tracked per channel.

Conclusion after architectural improvements of the R.E.A.L. architecture the processor load is 1.15 Gcycles/sec for basic speech service. So concluding, it very hard to get a DSP implementation of the Rake receiver using R.E.A.L. DSPs.

Low End (basic speech):
Number of downlink channels= 3,
Number of paths to be tracked= 3,
SF=16
Memory after despreading (1)
Memory size 5.63 Kbits
Bandwidth 4.5 Maccess/sec
memory before despreading (2)
Memory size = 40 Kbit
Bandwidth = 44 Maccess/sec
4 Designflow

4.1 introduction

The goal of the design flow is to provide a path from high level algorithm evaluation to a cycle accurate, bit true VHDL implementation of a VLIW A|RT processor capable of running the algorithm efficiently. The development process allows incremental refinement of individual functional units. The inputs of the design flow are:

- C, used to describe the behavior of the algorithm
- A|RT library, contains the behavioral models in C and VHDL of the standard functional units
- Simulink wrapper, used for compiling the C program, in such a way that it provides the correct interface to Simulink.
- Simulink model, describes the model of Rake receiver
-Pragma files, used to describe the functional units in the VLIW processor, and the number of instantiations.

The first step in the design flow is understanding the Rake algorithm. This is done using an existing WCDMA Mathworks Simulink model, and reading literature concerning CDMA and known Rake receiver implementations (see chapter 3). Matlab in combination with Simulink is chosen as simulation tool, because it provides built in data processing and analysis functions as well as extensive graphical capabilities. The Simulink models can be extended with user defined functions, written in C or Matlab.

Then next step was making a derived RAKE receiver model of the Mathworks WCDMA chain Simulink model.

The next step in the design flow was replacing Simulink (arithmetic and predefined) blocks by, user defined S-functions. The S-functions are dynamically linked library functions, created using a C program which describes the behaviour of the Rake algorithm. A Simulink wrapper must be added to the C function to supply a correct interface to Simulink.

When the Simulink system simulation is verified and correct, the C source is used to design a VLIW A|RT processor.
The behavior description in C of the algorithm is used to design a processor. The design of the processor is done using the tool A\|RT Designer from Adelante Technology / Frontier Design. The tool tries to map the C description on the a specific architecture. This architecture is next to the C description the second input of the tool. After mapping and scheduling the C program on the architecture, A\|RT Designer gives information on the number of cycles it needs to executed the program. The cycle count has to be optimized in such a way that the number of cycles is minimum. This can be done by modifying the architecture description of the processor. This is an iterative process. When the cycle count is optimal, RT^7 level VHDL can be generated of the architecture of the processor.

The use of A\|RT designer in the designflow fully optimizes the time sharing of hardware resources, to get a load balanced schedule and a more efficient use of the available resources.

The next step is synthesis flow and verify the generated processor on VHDL and netlist level with HDL input stimuli. The verified verilog netlist will provide estimation information about timing and area of the design.

This table gives a brief overview of primary tasks and tools

<table>
<thead>
<tr>
<th>Task</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>VHDL Simulation &amp; Co simulation</td>
<td>NC-VHDL, NC-Verilog, NcSim, Tempo.HDL_Flow</td>
</tr>
<tr>
<td>Verilog Simulation</td>
<td>NC-Verilog, Vnavigator</td>
</tr>
<tr>
<td>Power Simulation analysis</td>
<td>Cadence Build Gates, Diesel, wireload model</td>
</tr>
<tr>
<td>RTL synthesis</td>
<td>Cadence Build Gates</td>
</tr>
<tr>
<td>Algorithm development</td>
<td>Matlab, Simulink</td>
</tr>
<tr>
<td>Hardware/ software partitioning</td>
<td>A|RT designer</td>
</tr>
<tr>
<td>mapping to architecture</td>
<td></td>
</tr>
<tr>
<td>scheduling</td>
<td></td>
</tr>
</tbody>
</table>
4.2 Matlab/ Simulink

The first step in the design process was to get acquainted with the application, i.e. reading articles and trying to understand a MathWorks Simulink model of a wideband CDMA transmitter-receiver chain. (see figure 53)

Simulink provides built-in data processing and analysis functions as well as extensive graphics capabilities. The next step was writing a floating point C program, this code is re-used in designing the AIRT processor. In Simulink a design is represented as a (hierarchical) block diagram, with code in C or Matlab associated with each block. Simulink comes with a large library of predefined blocks, which can be extended by users who write their own so called S-functions. Since Simulink defines the block/ S-function interfacing mechanism, code can be easily be encapsulated and shared. The process of designing the system in this environment is done as follows. Initially the system is modelled at the behavioral level with all the signal processing being performed using floating point arithmetic and predefined Simulink blocks. Subsequently, all blocks are gradually refined. The first refinement step consists of replacing all the floating point data types and operations with finite precision processing employing limited word lengths (i.e., these Simulink blocks are replaced by custom S-functions). The fixed point C++ class library provided by AIRT library from Adelante Frontier Design is used to model the bit true implementation.

The proposed sub design flow is presented in figure 54.

4.2.1 Associated files

The organization of the files used in the design flow, associated with the for example the functional unit code generator, is presented in the next table.
<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGU.cxx</td>
<td>Functional unit implementation</td>
</tr>
<tr>
<td>CGU.h</td>
<td>Class implementation of registers</td>
</tr>
<tr>
<td>CGU.pra</td>
<td>Function unit architecture</td>
</tr>
<tr>
<td>CGU.obj.cxx</td>
<td>Simulink state information for S-function</td>
</tr>
<tr>
<td>CGU.wrap.cxx</td>
<td>Simulink wrapper</td>
</tr>
<tr>
<td>CGU_model.mdl</td>
<td>Simulink model</td>
</tr>
<tr>
<td>Makefile</td>
<td>GNU Makefile used for compiling and linking .mexglx</td>
</tr>
<tr>
<td>CGU.mexglx</td>
<td>Linux dynamic linked library for Simulink</td>
</tr>
<tr>
<td>CGU_e.vhd</td>
<td>VHDL entity</td>
</tr>
<tr>
<td>CGU_rtl_a.vhd</td>
<td>VHDL architecture</td>
</tr>
</tbody>
</table>

For each functional unit in AIRT Designer a behavior C description is made (CGU.cxx) and a description of the timing behavior (CGU.pra). The state information and the interface for the S functions in Simulink are describe in CGU_obj.cxx and CGU_wrap.cxx. The hierarchically Simulink structure of the model is describe in the CGU_model.mdl. The CGU_mexglx is the library that can be dynamically linked to the Simulink model.

The VHDL entity and architecture description of the functional unit is describe in the CGU_e.vhd and CGU_rtl_a.vhd.

![Simulink sub model of Rake receiver](image)

**Figure 55: Simulink sub model of Rake receiver**
4.2.2 S-Function

An S-function is a computer language description of a Simulink block. S-functions can be written in MATLAB, C, C++, Ada, or Fortran. C, C++ S-functions are compiled as library files using the Matlab mex utility or compiling as .DLL/Mexglx with a C compiler. As with other MEX-files, they are dynamically linked into MATLAB when needed.

S-functions use a special calling syntax that enables the interaction with the Simulink equation solvers. This interaction is very similar to the interaction that takes place between the solvers and built-in Simulink blocks. The form of an S-function is very general and can accommodate continuous, discrete, and hybrid systems.

After writing the S-function and placing its name in an S-Function block, it's possible to customize the user interface by using masking.

![Simulink customized interface](image)

Figure 56: Simulink customized interface
4.3 A|RT Designer

A|RT Designer is a software tool developed by Frontier Design/ Adelante Technology. This tool is utilized to implement an algorithm in digital synchronous hardware. More specifically, the tool assists in the translation of an algorithm written in a sub set of the programming language C into a processor or a processor like architecture described in VHDL. The processor is customized for the algorithm that has to be executed on this architecture.

The generated processor consists of a set of datapath resources, controlled by a VLIW (Very Long Instruction Word) type controller. This highly configurable controller architecture is scalable for parallelism as well as performance.

Figure 57: A|RT designer main gui

A|RT Designer facilitates the exploration of several alternative architectures, and allows through user interaction to determine the optimal architecture for the algorithm. The process of obtaining an optimal architecture is through an iterative process. Based on the feedback and the interactive possibilities of the tool. The designer can make trade offs analysis and gradually steer the tool towards the desired goal.

The starting point of A|RT Designer is an algorithm intended for implementation on an ASIC or on a FPGA. The algorithm should be expressed in a subset of C.

On top of this C subset, the tool supports most of the C++ constructs provided in A|RT library and in SystemC. With the fixed point data types and operators in these libraries the designer has complete control over the dimensions of the operators.

The final output can be an RT level VHDL, verilog netlist or a cycle true C++ model.

Figure 58 depicts the A|RT Designer flow. The same steps are visible in the main gui see figure 57. The steps made in the design flow are iterative, i.e. depending on the outcome of each step, e.g. cycle count, a previous step can be modified. The design process is repeated at that particular level where the modification takes place.
Figure 58: ART Designer designflow
4.3.1 Processor architecture template

ART Designer maps a C algorithm into a RT level hardware description. The architecture can be described as the collection of register files, the paths between them and the resources along these paths that pass, modify or store the data. This set of hardware execution units and busses is the datapath. These are managed by a controlling mechanism: the controller or the control path. The controller controls the functional units through a very low instruction word (VLIW).

The datapath and the controller are the two constituting parts of the architecture. Figure 59 shows an example of this architecture.

![Art Designer architectural model](image)

Figure 59: ART Designer architectural model

The datapath consists of a number of execution units and the busses that interconnect them. In most cases, the busses are just a connection between an output port of an execution unit and an input port of the same or another execution unit. The figure displays the central resources of each unit and it illustrates the VLIW aspect of the controller: every resource is controlled separately by a vertical partition of the controller.

The datapath is defined at a high level, in terms of functional units. The tool comes with a complete set of highly parameterized resources like adders, multipliers, ALUs and memories to accommodate virtually any design problem. Auxiliary resources like register files, multiplexors, tri state buffers and busses are all introduced automatically by the tool.

The VLIW processor is an architecture with a high level of parallelism. The parallelism can be seen both in instruction and data parallelism.

Instruction parallelism allows multiple instructions to be instantiated at a single clock cycle.

The data parallelism exploits the ability of executing the same instruction on different data at a single clock cycle.

The ART VLIW processor template consists of multiple independent functional units. Each unit is capable to execute a single instruction or multiple instructions.

Custom designed functional units can be added to the architecture to speed up the cycle count of the algorithm.

Datapath The datapath can be best described by paths it uses to transfer data from one register to another register and in terms of resources allocated along those paths. Figure 60 illustrates the transfer paths.

Any RT starts by reading data from one or more fields in different register files. It finishes by writing derived data into one or more fields in different register files. The top of the figure shows the registers being written to.

The functional units (core resources) are the functional hardware that performs the data transformations. These functional units use the data that is available in the assigned registers. The data produced by the
The architecture has besides a data path also a control path with a controller. All resources on the datapath are controlled by it as it organizes the data flow through the datapath.
4.3.2 Logic synthesis

A|RT designer is a tool that automatically translates an executable C program into RT (register transfer) level VHDL. At the RT level, the timing of a design is fixed at the resolution of clock cycles. The processor architecture design can be seen as a state machine in which the registers hold the system state, i.e. the RT level description specifies the next state of the state machine.

At this stage logic synthesis can be performed to design the combinational logic that will implement the next state function. If the input description is given in VHDL, the process is called VHDL synthesis. A handy property of VHDL synthesis is that the VHDL code that can be processed by the synthesis tools, is in principle independent of the target implementation, whether it is an application specific integrated circuit (ASIC) or a field programmable gate array (FPGA). Both types of implementations differ at the level of basic building blocks, i.e. standard cells. All available cells are part of a library. The VHDL synthesis tools do not need to know all details of library cells. What matters is the functionality and the delays associated to the propagation of the signals through the gates.

After logic synthesis, the design will consist of an interconnection of library cells, the netlist. The netlist needs to be processed by backend tools that are specific for the target implementation.

In the case of an ASIC, the backend tools will generate the layout of the entire chip by placing and routing the cells.

In this master thesis the Cadence BuildGates is used for logic synthesis.

In the synthesis flow the consists of the following steps:

- Creating a functional VHDL description, i.e. RT level description, of the processor architecture and verify its functionality using Nc Sim.
- Synthesizing the functional VHDL description into a gate level netlist composed of standard cells. This will provide timing and area constraints to drive the synthesis.
- Verification of functionality of the synthesized design by simulating in verilog.
- Performing timing analysis to determine whether the design met its design goals.
4.4 Behavioral synthesis

Behavioral specifications are inputs to behavioral synthesis tools. A behavioral specification is a description of the functionality of the design. This description consists of arithmetic and logical operations arranged in sequences along with branching structures, and loops. A behavioral specification expresses the functionality of a design in terms of operations and interactions with the environment. It does not specify what resources are allocated to perform the operations or the order or clock cycle in which individual operations should be performed. In VHDL terms, a behavioral specification uses a behavioral subset of VHDL, i.e. it uses sequential code in process statements. A behavioral synthesis tool can take a behavioral description of a design written in HDL and generate an RTL description of the design. In the Build RT level step in A]RT Designer the operations (for example, multiplication, addition, and comparison) in the algorithm are implemented using resources like a multiplier or an ALU of a hardware micro-architecture. The compiler schedules single algorithm steps into multiple hardware clock cycles and assigns them to the hardware resources available for execution. The output from the tool is an RTL representation of the design, describing all the interconnection of the registers and functional units that perform the arithmetic and logical operations on that data.

4.4.1 Behavioral Synthesis Concepts

In the design process, a synthesis tool generates a hardware design that is functionally equivalent to the input behavioral specification. Since timing is not explicitly specified in a behavioral model, there are timing differences between the behavioral input specification and the hardware specification that is generated. The functional behavioral for both specifications are the same. The behavioral description used as input by the synthesis tool is technology independent. To ensure cost-efficient hardware solutions, the synthesis tool must take into account the timing and area characteristics of the target technology library. For this reason, technology information is an input to the synthesis tool.

4.4.2 Behavioral limitations

All synthesis tools limit the abstraction level of the input description as well as the nature of the hardware generated. For example, recursive and dynamic data types cannot be directly supported in hardware, which is static by nature.

Although any specification can be expressed in a style suitable for behavioral synthesis, not all applications should be expressed this way. The strong points of behavioral synthesis are scheduling and resource sharing. To benefit from this, the application should allow for placement of operations in different clock steps to facilitate resource sharing, and choose operations that benefit from resource sharing and are not affected by the interconnect cost.

Until recently, there have been few, design tools that enable the designer to quickly evaluate architectural alternatives. Without the use of such tools, the time required to complete the implementation of one particular architecture typically precludes evaluation of alternate architectures, even if the current architecture is known to be sub-optimal.
4.5 VHDL / Verilog simulation

For the functional verification of the circuit a VHDL/Verilog simulator of cadence (NC VHDL) is used.

![Figure 62: Cadence NC VHDL](image)

```vhdl
ARCHITECTURE struct OF std_bench IS

COMPONENT hive_clockgen

BEGIN

GENERIC (

clk_low : time := 50 ns;

clk_high : time := 50 ns;

input_delay : time := 50 ns;

control_delay : natural = 3

);

PORT (

stepclk IN std_logic;

clk_button : OUT std_logic;

clock0 OUT std_logic;

clkimport : OUT std_logic;

clk_export : OUT std_logic;

rbtn_event_n : OUT std_logic;

rbtn_event : OUT std_logic

);

END COMPONENT;
```

©Eindhoven University of Technology 2003

53
4.6 Power estimations: DIESEL wire load model

Diesel is an acronym for Dissipation Estimation Software Extension for Logic simulation. As the name says, it provides existing logic simulators with additional functionality. Diesel basically keeps track of the instantaneous signal transitions that occur during a simulation. By combining this transition information with a one time library characterization, it determines the instantaneous supply current, and derivatives thereof. Currently, these derivatives are the total instantaneous and average power and energy per net and per hierarchical block and the activity numbers per hierarchical block.

4.6.1 Dissipation in CMOS circuits

This section describes static and dynamic power dissipation, and the method to calculate the power consumption dependent on the fanout load capacitance and switching frequency.

The environment of a CMOS cell can be modeled by associating a load capacitance with each output of the cell, and an input slope for each input. See figure 63. The output load consists of the capacitance associated with the net (wire capacitance) and the capacitance associated with each of the ports connected to this net (input load or fanin). The input slopes result from either primary input slopes or from other cells driving this cell. In CMOS circuits each cell consists of a PMOS and a NMOS block. If the output signal is stable, i.e., the signal value at the output of the cell does not change, only one of the blocks is conducting and thus no current flows from supply to ground.

![Figure 63: Power dissipation in a CMOS inverter cell](image)

Whenever the output has an event, i.e., the signal value at the output changes from zero to one or one to zero, the cell draws a current from the power supply. This current consists of three parts:

1. During the time at which both parts are conducting, a short circuit current flows through the PMOS and NMOS parts.
2. A dynamic load current that flows from \( v_{dd} \) through the PMOS to the capacitor for rising events, and from the capacitor through the NMOS to ground for falling events.
3. An internal dynamic load current that charges/discharges capacitances present within the cell.

For most circuits, the main part of the power dissipation is represented by dynamic load current (about 70-90%).

The currents only flow during the short period of time in which the output signal voltage changes. The current waveform heavily depends on the input slopes, the output load and the type of output signal value change (being either 0-1, 1-0), called the event type.

Besides the supply current, also the behavior of the output signal voltage depends on the input slopes and the output load. The rate of change of the output slope decreases with increasing load and decreasing input slope. And since the output is connected to other input cells, this in turn affects the input slopes of the cells driven by the cell’s output.
4.6.2 Static power dissipation

For most circuits, the main part of static power dissipation can be attributed to power leakage. This part of the power dissipation is independent of the switching frequency of the cell. Static power is dissipated in several ways. The largest percentage of static power result from source to drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called leakage power. The total static energy for a block of logic is the sum of the static power of each library element used in that block.

Leakage or static power typically represent 5-10% of the total dissipated power in a CMOS cell.

4.6.3 Dynamic power dissipation

Dynamic power dissipation is due to the switching of the logic cell, when one or more inputs of the circuit change their input values. The cell will dissipate dynamic power even if the input change does not result in a logic transition on the output. For current CMOS technologies, dynamic power dissipation represents the majority of the power consumption.

There are two major sources to dynamic power dissipation:

1. Switching power
   The switching power of a cell is the power dissipated by the charging and discharging the load capacitance connected to the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitance on the driving output. Since charging and discharging only occurs when the output of the cell changes logic state. The total switching power of the cell increases when the transition frequency increases. Switching power is the main source of power dissipation of an active CMOS cell.

2. Internal power
   Internal power is any power dissipated inside the cell. During switching, a circuit dissipates internal power by the charging or discharging of capacitances internal to the cell. Internal power also includes power dissipated by a momentary short circuit between the P and N transistors of a gate when the input voltage is in the range where both transistors will conduct. This is usually called the short circuit power. Short circuit power can be as much as 30% of the total power dissipated by the cell.

4.6.4 calculating power dissipation

To be able to make an accurate estimation of the power dissipation for a CMOS cell, the following factors have to be known:

- The static power dissipation
- The energy consumption for the rising and falling transitions
• The energy consumption when charging and discharging the fanout load (transition that causes a change on the output)

• The switching frequency

The total power dissipated by a CMOS cell library element at a given frequency is given by the following equation:

\[ P_{\text{diss}} = (E_{\text{rise}} + E_{\text{fall}} + (C_{\text{fanout}} \cdot V_{\text{dd}}^2)) \cdot F_{\text{switching}} + P_{\text{static}} \]  

(16)

where

- \( P_{\text{diss}} \) total power dissipation of the gate (\( \mu \)W)
- \( E_{\text{rise}} \) energy for a rising transition on the output (pJ)
- \( E_{\text{fall}} \) energy for a falling transition on the output (pJ)
- \( C_{\text{fanout}} \) output load capacitance, the number of loads (external inputs connected to the output of the cell) multiplied by the value for a standard load (pF)
- \( V_{\text{dd}} \) supply voltage
- \( F_{\text{switching}} \) Switching frequency of the transition (MHz)
- \( P_{\text{static}} \) Static power dissipation of the library cell (\( \mu \)W)

To calculate the power dissipation of more complex blocks, the switching frequency of each single cell has to be known. The switching frequency of the input for a specific cell is dependent on the switching frequency of the output of the other connected cells, and recursively, of the inputs and the input of the whole logic block.

By doing simulations with the RTL code, the simulator can gather information about switching of the input and output of the whole block. This switching information can then be used to estimate the average power dissipation of the specific block with that particular input data. The switching information is stored in a SAIF file (Switching Activity Interchange Format) which the power estimation and synthesis tools can be used to estimate and optimize for power.
4.6.5 Wire load models

Wire load models are defined in the technology libraries and are used to estimate the net capacitance in a design. The estimated net capacitance is then used to estimate the net delays as well as parts of the cell delays (transition delays and slope delays). In this master thesis a CMOS12 wire load model is used.

After synthesis a hierarchical model of area usage of the design is obtained. Each level represents a set of sub cells. The estimated capacitance of a net is dependent on:

- the number of inputs to a net
- the total summation of the cell areas in which the net is present.

To obtain the capacitance a top down search is executed in the hierarchical model. At the hierarchical level where the net is still present in one sub block, i.e. not divided into two sub blocks, the total area of all cells, present at that level, is calculated. The area of each standard cell is defined in the standard cell library (core lib). After summation over all cell areas, the total area is the input to a look up table. The table has been subdivided in classes depending on the area.

Example:

"50K" {
    wire_cap {
        table.axis1 = "load" {
            1.0;
            2.0;
            15.0;
            16.0;
        }
        table.data {
            0.0021;
            0.0049;
            0.0686;
            0.0735;
        }
    }
}

In this example a part of the look up table is presented, these values are valid to a total value of 50k gates. The load (number of inputs to a net) determines next the total capacitance of that particular net. The content of the lookup table is based on a statistical data which is regular updated. When a circuit design is actual made, the capacitance values are measured and compared with the estimated values from the model. With this new information the model is updated, and therefore becomes more accurate.

The estimated capacitance values are becoming more inaccurate when the length of a net increases (global nets, such as clock and reset). Because the number of cells where the net is presents also increases.

Custom wire load models may be created for larger designs, designs with unusual aspect ratios, highly congested or highly sparse designs, or if the physical designs differs drastically from the logical design.

The flow for creating custom wire load models may be to perform initial synthesis, floorplan the design for more accurate wire length estimations, back annotate the wire capacitances.

Operating conditions are defined in the technology libraries and are used to define environmental conditions for operating temperature, supply voltage, and manufacturing process. To ensure that acceptable
performance levels are maintained over a range of operating conditions. In this master thesis the worst case military conditions are used for timing analysis.

Library : corelib
release v5.0-s005 nov6 2002

<table>
<thead>
<tr>
<th>PVT</th>
<th>Operating condition</th>
<th>process</th>
<th>temperature</th>
<th>voltage</th>
<th>OC type</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>WC MIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
<tr>
<td>typ</td>
<td>WC MIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
<tr>
<td>min</td>
<td>WC MIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
</tbody>
</table>

4.6.6 Back annotation of wire capacitance

As was indicated in the previous section, the dynamic load current is the major contributor to the overall power behavior of a cell. This contribution scales linearly with increasing load capacitance according to:

$$ E = \frac{1}{2} C V_{dd}^2 $$

where

$ V_{dd} $ supply voltage

$ C $ the sum of the wire capacitance and the fanin capacitance of the ports connected to the net

The fanin capacitances are available in the power view of each cell. They do not depend on the final layout of the circuit. The wire capacitance does depend on this final layout. Thus, to obtain realistic power figures, the wire capacitance should be specified for each net. This is called back annotation.

There are two ways to obtain wire capacitances, by extraction and by estimation. The first method assumes that layout is available for the circuit. Using a parasitic extraction tool, a list of net name versus capacitance value is obtained. The second method calculates a capacitance value for each net, based on some properties of the net and/or the circuit, according to a user specified symbolic expression.

4.6.7 Back annotation of timing

Besides the impact of wire capacitance on the power figures, the input slopes and the output load also influence the delay of each cell. Thus to have realistic timing during logic simulation, the wire capacitances have to be taken into account for correct determination of the pin to pin delay for each cell.

4.6.8 Test bench dependence

The estimated power figures heavily depend on the set of vectors (the test bench) that is simulated. Therefore a good choice of vectors is important to obtain realistic power figures.

The test bench used to obtain the power estimations of this master thesis, is the behavior of the Rake receiver (4 fingers and 1 maximum ratio combiner).
5 Architecture

5.1 Introduction

This chapter deals with the process of coming to a scalable processor architecture which is power and area efficient. The processor architecture consists of a combination of generic and dedicated functional units. Dedicated units to speed up the bit level manipulations and generic to be able to map multiple algorithms on the processor.

The first chapter discusses the system trade-offs.

5.2 System architecture design trade-offs

There are different rake receiver implementations possible (see chapter 3), each with their own pros and cons. The different solutions can be divided into two subclasses: software and hardware solutions:

- Programmable solutions are limited to a fixed cycle budget to perform a specific mapping of an algorithm. The solutions have in that sense a fixed upper boundary. It is on the other hand possible to map multiple algorithms on the same core. So it that sense these solutions are flexible.
- Hardware solutions, like ASICs, are specially designed to efficiently map an particular algorithm. These solutions are only capable performing one specific algorithm. The flexibility vs. efficient implementation is opposite.

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Cost</th>
<th>Power</th>
<th>Flexibility</th>
<th>Design effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DSP</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>FPGA</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>General purpose processor</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>AIRT processor</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 1: Scalability

The goal of this master thesis was to design an architecture that has the benefits of both solutions. Hence the architecture has to be generic, to run multiple (domain specific) algorithms, but at the same time still maintain an efficient mapping of these algorithms, in terms of minimizing the number of cycles.

The following issues in the design process of the architecture are important:

- Scalability (time / space)
- Memory usage (size / bandwidth)
- Exploiting parallelism
- Load balance

5.2.1 Scalability of the design

Scalability is an important consideration in system development. The processor architecture should scale (power, area, ...) with the number of rake receivers mapped to one processor, and the architecture should not be restricted to a upperbound on the number of fingers. This implies that an architecture with common
resources used, is not feasible. A common resource creates a bottleneck. For example a common memory has a limited access bandwidth and limits the number of processes that can access the memory in a certain amount of time.

![Common memory resource](image)

Figure 65: Common memory resource

A distributed approach can provide better system scalability and exploit better the data parallelism of the algorithm. The operations in the kernel of the Rake algorithm only use local data, the so-called locality of reference is preserved.

![Distributed memory resources](image)

Figure 66: Distributed memory resources

To efficiently map the Rake algorithm, the data parallelism has to be exploited.

- The same operations are applied to multiple functional units
  Multiple Rake receivers are mapped onto different functional units (space diversity).
- Conceptually different operations are performed at the same time on different functional units (load balance).
5.2.2 memory usage balance

The concept of Rake receiver is that multi paths belonging to the same transmitter have to combined to one stronger signal (see chapter 2.5). To time align the signals received from different multi paths, the early arriving signals have to be delayed until all multi path components have arrived. This delay difference is also called delayspread. The delayspread is expressed as a multiple of the chip time \( T_c \) or chip period.

The maximum delay spread depends on the mulitpath propagation and soft handover.

It is assumed that:

\[
\text{Maximum delayspread} = 256 \cdot T_c
\]

where \( T_c \) = chip period

The UMTS chip rate is 3.84 Mchips/sec so:

\[
\text{Maximum delayspread} = \frac{256 \text{chips}}{3.84 \text{Mchips/sec}} = 67 \mu \text{sec}
\]

The design of the architecture has to be provided with a memory which is sufficiently large to implement this delay buffer. The time alignment can be done at chip or symbol level. This choice influences the location of the memory:

1. Before despreading
2. After despreading
3. In maximum ratio combiner

When the alignment is done at chip level the memory is placed before correlation. The chips are read from memory with different offsets for each multipath. The chips alignment is done by reading from memory with different address offsets.

When the alignment is done at symbol level, the memory is placed after the despreading operation or in the Maximum ratio combiner. Each finger is despreading the incoming chips with a different delay offset, i.e. different start time for correlation and different code offset. The depreaded symbols are stored in memory for maximum ratio combining.

The different approaches will be discussed in terms of access bandwidth and memory size.
Memory before despreading  The first implementation is placing the alignment memory as a single memory bank before the despreading function, like the FlexRake and DSP solutions.

Figure 67: Memory alignment before despreading, FlexRake implementation

The key idea of this solution is to first do the time alignment (at chip level) and then process the chips further. The memory bank stores the incoming oversampled chips.

The address of the memory has two fields. The first field specifies the chip location and second field specifies the sample to be selected for that chip. These address fields are controlled by the searcher and tracker, i.e. channel estimation/ tracking/ synchronization. For each path there is a separate address generated and the selected sample is send to the corresponding finger. By selecting the chips for each multipath from specific locations in the memory, the time alignment is achieved. The memory serves as a chip selector, by selecting a sample corresponding to a chip, depending upon the information received from the path synchronization. The memory address offset adjustment can also be done by changing the first field of the address for a specific multi path.

The implications on the hardware requirements will be discussed for this solution. The samples from memory which are delivered to the Rake receivers, from one base station, are time aligned. The same scrambling code can be used for all Rake receivers. The same holds for fingers receiving the same downlink channel, the channelization code generator can be shared.

The total memory size is dependent on the maximum delay spread, expressed in chips, for time alignment. Further it depends on the oversampling rate of the incoming samples, i.e. each (oversampled) sample has to stored in memory. The memory needs to store a complex sample, I and Q couple, this explains the factor of two.

The memory size can be written as:

\[ \text{Memory size [bits]} = O_s \cdot D_c \cdot \text{chipwidth} \cdot 2 \]  

where

- \( O_s \) = oversampling rate
- \( D_c \) = maximum delay spread
- \( \text{chipwidth} \) = the number of bits to represent one real chip

The memory access bandwidth is dependent on the number of write and read accesses to the single memory bank. Each chip period \( O_s \) samples are written to memory and for each finger a chip is read for despreading. For synchronization of each finger an addition 2 chips are used (early-late synchronization).

\[ \text{Bandwidth [Maccess/sec]} = R_c \cdot (O_s + \text{number of fingers} \cdot (1 + 2)) \]  

\( R_c \) = symbol rate

(18)

(19)
Memory after despreading  The second implementation is placing the alignment memory in the finger after the despreading operation but before the maximum ratio combination. In this way the alignment is done at symbol level. There are two options that can be chosen with respect to the control of memory.  

1. Firstly a single memory bank solution instead of separate distributed delay memories for each Rake receiver. This implies one common address generator. The memory can split into different parts, each part corresponds to one Rake receiver. The symbols coming from all fingers corresponding to one Rake receiver, are stored in the corresponding part that is reserved. In this way all received symbols corresponding to a transmitted symbol are stored in 4 consecutive memory locations. With each completion of the finger that is tracking the longest path, the maximum ratio combination reads the symbols from four consecutive memory locations and combines them. The memory size for this solution is reduced by a factor of the spreading factor and oversampling ratio compared to the memory before despreading. But the memory size is also increased because the stored symbols have more bit precision as stored chips.

\[
\text{Total memory size [bits]} = \frac{D_c}{SF_{\text{min}}} \cdot (\text{number of fingers}) \cdot \text{symbolwidth} \cdot 2 \quad (20)
\]

where

- \(D_c\) = chip time
- \(SF_{\text{min}}\) = minimum spread factor
- \(\text{symbolwidth}\) = number of bits to represent one real symbol
- factor of 2 because the symbols are complex (I & Q).

The memory access bandwidth is linearly dependent on the number of fingers and the spreading factor. All Rake receivers use the same memory, thus the memory access increases.

\[
\text{Bandwidth [Maccess/sec]} = \frac{R_c}{SF_{\text{min}}} \cdot \left(\frac{\text{number of fingers}}{\text{Rake receiver}}\right) \cdot (\text{number of rake receivers}) \quad (21)
\]

where

- \(R_c\) = chip rate
- \(SF_{\text{min}}\) = minimum spreading factor

Separate memories for I and Q symbols.
2. Another solution is to distribute the memory in separate blocks for each finger. The control of the memory access can then also be distributed, hence reducing the access bandwidth to the memory.

The memory size is the same as the first solution, so:

\[
Total \ memory \ size \ [\text{bits}] = \frac{D_c}{SF_{\text{min}}} \cdot (\text{number of fingers}) \cdot \text{symbol width} \cdot 2
\]  

(22)

where

- \(D_c\) = chip time
- \(SF_{\text{min}}\) = minimum spread factor
- \(\text{symbol width}\) = number of bits to represent one real symbol

factor of 2 because the symbols are complex (I & Q).

The memory access bandwidth is also linearly dependent on the number of fingers, but because of the distributed approach the memory access bandwidth is reduced.

\[
\text{Bandwidth} \ [\text{Maccess/sec}] = \frac{R_c}{SF_{\text{min}}} \cdot \left( \frac{\text{number of fingers}}{\text{Rake receiver}} \right)
\]  

(23)

where

- \(R_c\) = chiprate
- \(SF_{\text{min}}\) = minimum spreading factor

Separate memories for I and Q symbols.
Memory in maximum ratio combiner  The last possible implementation of the memory alignment is the place the memory in the maximum ratio combiner. The maximum ratio combiner combines the symbols from each active finger into a single stronger symbol (see figure 27). Since the fingers may be separated in time with up to 512 Tc=134 µs, the memory needs to buffer the partial accumulated symbols until all active fingers have arrived. The maximum delay between the same symbols from different fingers is 128 symbols (SF=4).

The maximum ratio combiner stores the partial accumulated symbols in a circular buffer, which is implemented in a similar way to the stream buffer in the FlexRake solution. At the moment when the maximum ratio combiner receives a symbol, it first reads the partially accumulated/combined symbol from memory. The received symbol is accumulated with the symbol from memory. The newly combined symbol is written to the memory at the same location as the partially combined symbol is read. If the received symbol was from the finger with the largest multipath delay, the accumulated value is sent the output port, and the memory location is cleared.

![Figure 70: Memory alignment in Maximum ratio combiner](image)

The memory size will be further reduced because the symbols of the multipath components are partially accumulated, i.e. only one memory location per combined symbol.

\[
\text{Total memory size [bits]} = \frac{D_c}{SF_{\text{min}}} \cdot \text{symbolwidth} \cdot 2 \cdot (\text{number of rake receivers})
\]  

(24)

where

- \(D_c\) = maximum delay spread
- \(T_c\) = chip time
- \(SF_{\text{min}}\) = minimum spread factor
- \(\text{symbolwidth}\) = the number of bits to represent one real symbol

The memory access bandwidth has the same properties as the memory after despreading solution, so multiple Rake receivers mapped to one physical memory or distributed over multiple memories.

\[
\text{Bandwidth [Maccess/sec]} = \frac{R_c}{SF_{\text{min}}} \cdot \frac{\text{number of fingers}}{\text{Rake receiver}} \cdot (\text{number of rake receivers})
\]

(25)

where

- \(R_c\) = chip rate

©Eindhoven University of Technology 2003
SF_{min} = minimum spreading factor
The number of Rake receivers is 1 in a fully distributed approach.
Comparison of different alignment implementations. In figure 71 the comparison is presented with respect to memory size. In figure 72 the comparison is presented with respect to the memory bandwidth.

\[ O_s = 4 \]
\[ D_c = 512 \text{ chips} \]
\[ R_c = 3.84 \times 10^6 \text{ chips/sec} \]
\[ \text{chipwidth} = 8 \text{ bit} \]
\[ \text{symbolwidth} = 16 \text{ bit} \]
\[ SF_{\text{worst case}} = 4 \]
\[ SF_{\text{best case}} = 256 \]
\[ \#\text{fingers/ Rake receiver}_{\text{worst case}} = 6 \]
\[ \#\text{fingers/ Rake receiver}_{\text{best case}} = 3 \]

![Figure 71: Memory size vs. rake receivers](image)

©Eindhoven University of Technology 2003
In this thesis the assumption was made to have a scalable architecture, this excludes the possibility of option of memory in before correlation. In this solution the memory and code generator is a common unit and gives an upper boundary on the number of fingers that can be mapped. Further the use of common units introduce at layout level more wire length than solutions where the data is kept locally.

From the two resulting scalable / distributed solutions the implementation with the memory in the maximum ratio combiner is choosen for implementation, because this solution uses the available memory locations more efficiently.
5.3 Exploiting parallelism

There are two basic techniques that can be used to increase the processor performance:

- Implement the processor in a faster technology.
- Performing more operations in parallel.

It has become apparent that more parallelism must be exploited in order to keep up with the increasing system performance. There are a wide variety of architectures known, that attempt to exploit the parallelism available in the application programs. For example, pipelined processors and multiple instruction issuing processors, such as superscalar and VLIW machines, exploit the instruction level parallelism. In contrast, shared memory multiprocessors typically exploit parallelism by distributing entire loop iterations to different processors.

5.3.1 Limitations to parallelism

The parallelism available in an application is limited by its dependences. A dependence between two operations is a conflict that prevents the operations from executing concurrently. Dependences can be categorized into three types:

1. Resource
2. Control
3. Data

A resource dependence exists between two operations when they both need to use the same physical resource at the same time. These dependencies are a physical limitation of the actual machine on which a program is to be run.

A control dependence occurs when an operation should be executed only if a previous operation produces a certain value. For instance, a conditional operation produces a control dependence to a different operation if the second is to be executed only if the condition evaluates to a specified value. In the Rake algorithm the conditional statements are frequently used (see figure 73).

```
local_ACC_finger1 += despread_chip1;
if (local_SF_finger1 == 0) {
    Symbol_finger1 = local_ACC_finger1;
    local_ACC_finger1 = 0;
    address = delay_finger1 + counter;
    mrc_symbol_finger1 = MRC[address];
    MRC[address] = mrc_symbol_finger1 + Symbol_finger1;
}
```

Figure 73: Example of conditional code in Rake algorithm

Data dependencies, also known as hazards, are read-write conflicts between two operations in which they both access the same storage location, such as a register or location in memory.
In a flow dependence (read-after-write hazard), one operation needs a value generated by a previous operation before the latter operation can begin executing. An output dependence (write-after-write hazard) occurs between two operations when they both write to the same storage location. The correct ordering of these operations is required to ensure that any intervening operations that read this location obtain the correct value before it is overwritten by the second operation. An antidependence (write-after-read hazard) exists when a later operation may overwrite a value still waiting to be read by an earlier operation. Both antidependences and output dependence occur when variable names and registers are reused by the programmer or by the compiler to reduce the number of unique memory locations referenced by a program. They can be eliminated by renaming variables so that a unique value of a variable has a unique name. The cost of this renaming is a potentially large increase in the memory requirements of the program.

5.3.2 Control dependency: Reduce sequential bottlenecks

From previous chapters can be concluded that:

- There exists parallelism in the baseband Rake processing algorithm.
- A distributed approach has the preference to an architecture with common functional units.
- The available parallelism has to be exploited to speed up the application program.

The consequences of mapping the Rake algorithm on an AIRT processor with a SIMD\(^8\) VLIW\(^9\) architecture will be discussed in terms of dependencies:

- The resource dependence is a physical boundary to the architecture. The size of processor has to be as small as possible.
- The data dependencies are solved by the scheduler of AIRT Designer.
- The control dependence is highly dominant in the Rake algorithm, this causes that the algorithm is executed in a sequential manner.

\(^8\)Single Instruction Multiple Data
\(^9\)Very Long Instruction Word
Rake algorithm mapped on a VLIW type architecture

The A|RT processor template consist of a number of simple heterogeneous processing elements connected to small memories and register files. A single control unit fetches one (very long) instruction word which is then decoded and broadcast to all functional units, and causes a data element to be fetched from register/memory or input port (see chapter A|RT processor template).

Each functional unit then executes the instruction on the data it has been given. At the end of the instruction the resulting data is written back to register/memory or output port. The processing of these VLIW instructions is synchronous to the central control unit. Multiple functional units can be initialized in parallel (instruction parallelism) to increase the performance of the processor.

In a single controller architecture the conditional statements are executed in a sequential manner, this sequential behavior forms the bottleneck in exploiting the parallelism in the Rake algorithm.

In a distributed implementation of the Rake algorithm, all rake fingers and maximum ratio combiners can
be modelled as independent processes. Each process has its own (branch) controller. The processors are mutually connected via a global interconnect (see figure 76).

Figure 76: MIMD VLIW architecture

It can be seen as a pure MIMD (multiple instruction, multiple data) application, in which each Rake finger performs a correlation with a completely different code and a different spreading factor.

At the other end the Rake algorithm can be seen as a data-parallel application that can run in a pseudo SIMD mode, with every Rake finger performing the same set of operations on its portion of a distributed data structure. This is however not directly possible because of the conditional behavior of the Rake fingers, e.g. Conditional dumping of accumulated symbols for each finger. The conditional operations must still be under local control of each Rake finger.

The resulting processor architecture model is one in which each Rake finger executes the same program instructions, but has some extended flexibility in local control flow.
Concept of conditional operations in data path  The concept of the conditional accumulation of the Rake fingers is shown in figure 77.

```
void conditionaltest( const Int<16> sf, Int<16> & out) {
    #pragma OUT out
    Int<16> local_sf=sf;
    while(1) {
        local_sf -= 1;
        if (local_sf == 0) {
            local_sf[1] == sf;
            out = local_sf;
        } //if
    } //while
} //conditionaltest
```

The schedule of the program on a single controller architecture, without placing the control information locally in the datapath, is as follows:

The while loop can be seen as a rectangle over the cycles numbers 4 till 10. In the body of the while loop a conditional statement (if statement) is executed in cycles number 9 and 10. The cycles 7 and 8 represent the branch delay.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ALU</th>
<th>ROMCONTROL</th>
<th>INPORT</th>
<th>OUTPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>&lt; 0 &gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>&lt; 1 &gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Sub Pass</td>
<td>&lt; 0 &gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Pass</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>Write</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The condition can be moved from the controller into the datapath, the control information is part of the data.
path. The functional unit ALU is extended with an extra data and control input (see figure).

Figure 79: **Condition locally in the datapath**

This abstract example represents the updating of the spreading factor for each finger. Variable local_sf is mapped to register R31 and variable sf to register R2. In register R4 is constant value 1 is stored.
Conditional Rake finger  This example of conditional updating the local spreading factor can be extended to a complete finger. In figure 80 multiple Rake fingers are time multiplexed on the same ALU.

Figure 80: Control data in data path of ALU

where:

<table>
<thead>
<tr>
<th>ALU1</th>
<th>ALU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 condition register</td>
<td>condition register</td>
</tr>
<tr>
<td>R2 spreading factor</td>
<td>0</td>
</tr>
<tr>
<td>R3 subtracted spreading factor values</td>
<td>accumulated finger values</td>
</tr>
<tr>
<td>R4 1</td>
<td>incoming despreaded chips</td>
</tr>
</tbody>
</table>

Table 2: ALU registers usage

The incoming despreaded samples (ALU 2 $\rightarrow$ R4) are accumulated with the local accumulation value of each finger (ALU 2 $\rightarrow$ R31...R34). In ALU 2 $\rightarrow$ R31 the accumulation value of finger 1 is stored and in ALU 2 $\rightarrow$ R32 the value for finger 2 etc.

ALU 2 $\rightarrow$ R1 is a control register that controls the selection of one input of the ALU 2. This register is normally 1, i.e. the inputs of ALU 2 are register R4 and one register of the set R31...R34.

ALU 1 controls the updating of the spreading factor for each finger in the Rake receiver, and sets a flag when the output of the ALU is zero. ALU 1 subtracts the registers R31...R34 with the value 1, each time a chip is accumulated in ALU 2. When the result of the subtraction is zero, i.e. R1 is zero, the next time not register R31...R34 is selected but R2 (spreading factor of the channel).

Register ALU 2 $\rightarrow$ R2 (zero) is also selected when register ALU 2 $\rightarrow$ R1 is zero, this means the local accumulation is dumped to a symbol and the register is resetted to zero.

In next figure a timing diagram is presented of the conditional operations. In the first chip period finger 3 completes the accumulation over a spreading factor. The symbol value of finger 3 is send to the maximum ratio combiner (not shown in figure). The next chip period the local accumulation register of finger 2 is initialized to zero. In chip period 2 finger 4 completes the accumulation.

Conditional maximum ratio combining In the single threaded solution the maximum ratio combining operation is a conditional operation, executed only at the end of the correlation period. This means that also the memory access, for coherent combining, is conditional.
When the condition is placed in the data path instead of executed as a conditional branch operation, the memory is accessed every chip. The behavior of the maximum ratio combining remains the same.

In the next section of pseudo C code the maximum ratio combining operation is presented:

```c
MRC:
if {local_SSF_finger1 == 0) {
    address = delay_finger1 + counter;
    mrc_symbol = MRC[address];
    MRC[address] = mrc_symbol + acc_finger1;
}
if (local_SSF_finger2 == 0) {
    address = delay_finger2 + counter;
    mrc_symbol = MRC[address];
    MRC[address] = mrc_symbol + acc_finger2;
}
......
if (local_SSF_fingerN == 0) {
    address = delay_fingerN + counter;
    mrc_symbol = MRC[address];
    MRC[address] = mrc_symbol + acc_fingerN;
}
```

The maximum ratio combining operation is repeated for all fingers (N). The schedule shows a highly serialized operations, multiple Rake receiver have to scheduled sequentially.

A solution to exploit the data parallelism that exist in the algorithm, is to place the condition (local_SSF_fingerX == 0?) in the data path, the same as in updating the spreading factor. This implies that for every chip a maximum ratio combining operation has to take place, but simplifies the schedule and allows executing multiple Rake receivers in parallel. The bandwidth of the memory of one Rake receiver is as a result increased.

The innerloop of the schedule is of length of the maximum number of fingers allocated to one Rake receiver (m). In this approach no communication takes places between the functional units and the controller. Multiple Rake receivers can be placed in parallel, under the control of the same instruction word. The fingers allocated to one Rake receiver are time multiplexed in this schedule, this reduces the area size.
if (local_SF_finger1 == 0) {
    address = delay_finger1 + counter;
    mrc_symbol = MRC[address];
    MRC[address] = mrc_symbol + acc_finger1;
}

Figure 82: Serial schedule of MRC operations

Figure 83: Parallel schedule of MRC operations
5.3.3 Load balance

It's important that each functional unit is occupied as much as possible. The Rake fingers assigned to one Rake receiver are processed sequentially. The functional units being time multiplexed for performing each of the operations. This makes sure that each functional unit is efficiently used.

The bottleneck of the Rake algorithm implemented, with the memory alignment in the maximum ratio combiner, is in the memory access used for the combining. The number of processor cycles is linearly dependent on the number of fingers assigned to one Rake receiver.

On the other hand multiple Rake receivers can operate in parallel, computing in space. The Rake receivers are then assigned to other functional units.

This combination of time multiplexed and space multiplexed operations increases the computational efficiency.
5.4 processor architecture

Figure 84: Rake processor
5.5 Benchmark

This chapter presents the final power and area estimations and the operating condition used in the synthesis flow.

5.5.1 Operating conditions

Library: corelib
release v5.0-s005 nov6 2002

<table>
<thead>
<tr>
<th>PVT</th>
<th>Operating condition</th>
<th>process</th>
<th>temperature</th>
<th>voltage</th>
<th>OC type</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>WCMIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
<tr>
<td>typ</td>
<td>WCMIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
<tr>
<td>min</td>
<td>WCMIL</td>
<td>1.50</td>
<td>125.00</td>
<td>1.10</td>
<td>balanced tree</td>
</tr>
</tbody>
</table>

5.5.2 Power estimation

The power estimation has been done on the verilog netlist with the functional testbench generated by A|RT designer and random input data. The clock rate used in the estimation is 100 MHz.

Diesel version: 2.8
Build for: NC-VERILOG v03.30.s003
Characterization: CMOS12 core library
Characterization: amdc_cmos12.lib/amdc_fdsram_512x32x16
ED&T Wiremodel: YES
ED&T Wire area: 0.000000
Wire cap. includes fanin: NO
Delay back annotation: NO

Process name: CMOS12
Char. temperature: 25.00 [Celsius]
Char. supply voltage: 1.200 [Volts]
Number of library cells: 133
Default slope rise: 352.0p [seconds]
Default slope fall: 352.0p [seconds]
Evaluation time interval: 0.14.54 µ seconds

Total initialization energy (X1 as 01 and X0 as 10) = 388.18p [Joule]
Total normal dissipated energy (only 01 and 10) = 40.22n [Joule]
Total input drive energy (not dissipated in cells) = 56.25n [Joule]
Total internal energy (only 01 and 10) = 31.33p [Joule]

Average normal dissipated power (only 01 and 10) = 2.77m [Watt]
Average input drive power (not dissipated in cells) = 3.87m [Watt]
Average internal power (only 01 and 10) = 2.15u [Watt]

Total average power (only 01 and 10) = 6.64m [Watt]
### 5.5.3 Area

<table>
<thead>
<tr>
<th>Block report for module 'hive_rake'</th>
<th>Current Module</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of combinational instances</td>
<td>20</td>
<td>9899</td>
</tr>
<tr>
<td>Number of noncombinational instances</td>
<td>0</td>
<td>2126</td>
</tr>
<tr>
<td>Number of hierarchical instances</td>
<td>137</td>
<td>419</td>
</tr>
<tr>
<td>Number of blackbox instances</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total number of instances</td>
<td>157</td>
<td>12444</td>
</tr>
<tr>
<td>Area of combinational cells</td>
<td>304.597</td>
<td>126942.399</td>
</tr>
<tr>
<td>Area of non-combinational cells</td>
<td>0.000</td>
<td>60039.943</td>
</tr>
<tr>
<td>Total cell area</td>
<td>304.597</td>
<td>186982.342</td>
</tr>
<tr>
<td>Number of nets</td>
<td>2319</td>
<td>14567</td>
</tr>
<tr>
<td>Area of nets</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Total area</td>
<td>304.597</td>
<td>186982.342</td>
</tr>
</tbody>
</table>

![Pie chart area usage](image)

Normal IC designs have a 80% effective usage of combinational area, due to decoupling capacitance, therefore the estimated area of 0.18 mm² will after correction ±0.20 mm².
### 5.5.4 Timing report

<table>
<thead>
<tr>
<th>Report</th>
<th>report.timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>hive.rake</td>
</tr>
<tr>
<td>Timing</td>
<td>EARLY / LATE</td>
</tr>
<tr>
<td>Slew Propagation</td>
<td>WORST</td>
</tr>
<tr>
<td>Operating Condition</td>
<td>WCMI</td>
</tr>
<tr>
<td>PVT Mode</td>
<td>max</td>
</tr>
<tr>
<td>Tree Type</td>
<td>balanced</td>
</tr>
<tr>
<td>Process</td>
<td>1.500</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.100</td>
</tr>
<tr>
<td>Temperature</td>
<td>125.000</td>
</tr>
<tr>
<td>time unit</td>
<td>1.000 ns</td>
</tr>
<tr>
<td>capacitance unit</td>
<td>1.000 pF</td>
</tr>
<tr>
<td>resistance unit</td>
<td>1.000 kOhm</td>
</tr>
</tbody>
</table>

#### Early

Path 1: MET Hold Check with Pin reg_in2.COMPARE.1/reg_u_1/qreg_1/CP

Endpoint: reg_in2.COMPARE.1/reg_u_1/qreg_1/D (v) checked with leading edge of 'clk'

Beginpoint: reg_in2.COMPARE.1/reg_u_1/qreg_1/Q (v) triggered by leading edge of 'clk'

<table>
<thead>
<tr>
<th>Other End Arrival Time</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hold</td>
<td>- 0.058</td>
</tr>
<tr>
<td>Phase Shift</td>
<td>+ 0.000</td>
</tr>
<tr>
<td>Required Time</td>
<td>- 0.058</td>
</tr>
<tr>
<td>Arrival Time</td>
<td>0.298</td>
</tr>
<tr>
<td>Slack Time</td>
<td>0.356</td>
</tr>
</tbody>
</table>

Clock Rise Edge = 0.000

Beginpoint Arrival Time = 0.000

#### Late

Path 1: MET Setup Check with Pin u.urom.ir/qr_reg_325/CP

Endpoint: u.urom.ir/qr_reg_325/D (v) checked with leading edge of 'clk'

Beginpoint: rst.a (v) triggered by leading edge of 'clk'

<table>
<thead>
<tr>
<th>Other End Arrival Time</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>- 0.103</td>
</tr>
<tr>
<td>Phase Shift</td>
<td>+ 10.000</td>
</tr>
<tr>
<td>Required Time</td>
<td>9.897</td>
</tr>
<tr>
<td>Arrival Time</td>
<td>- 8.458</td>
</tr>
<tr>
<td>Slack Time</td>
<td>1.439</td>
</tr>
</tbody>
</table>

Clock Rise Edge = 0.000

Input Delay = + 1.000

Drive Adjustment = + 2.324

Beginpoint Arrival Time = 3.324
5.5.5 Conclusions

In this master thesis an efficient implementation of a scalable WCDMA Rake receiver is presented. By placing the alignment memory in the maximum ratio combiner the architecture becomes scalable and memory bandwidth decreases. The total memory size is linear dependent on the spreading factor and number of Rake receivers.

For low end applications, number of Rake receivers \( i \leq 8 \), the total memory size is smaller compared to the memory before correlation.

For basestation applications, number of Rake receivers \( i \geq 100 \), the scalability is important, i.e. the bandwidth to the central memory is not sufficient.

A distributed approach with local memory is in both scenarios a better solution.

The exploiting of the parallelism of the Rake algorithm is in a single threaded solution bounded on the conditional branch instructions. The solution is to place the condition locally in the datapath. This ensures that multiple Rake receivers can be scheduled fully parallel (computing in space). The Rake receivers are assigned to other functional units.

To save area size, Rake fingers assigned to one Rake receiver are processed sequentially (space / time consideration). The functional units being time multiplexed for performing each of the operations.

This combination of time multiplexed and space multiplexed operations increases the computational efficiency.

The total design consumes

\[
6.6 \text{ mW} / 0.2 \text{ mm}^2 / 100 \text{ MHz} / 1.2V = 0.3 \text{ mW} / \text{ mm}^2 / \text{ MHz}
\]
References


## Index

**Acquisition phase**
- Matched filter, 24
- Serial/ parallel synchronization, 24
- Sliding/ active correlator, 24

**Anti jamming capability**, 7

**Back annotation**, 58

**CDMA**, 2
- Code division multiple access, 2
- CDMA2000, 2
- Conventional Rake receiver, 28

**Direct sequence**, 2, 3
- Direct sequence spread spectrum, 2
- DLL, Delay locked loop, 25
- DS, Direct sequence, 2

**FDMA**, Frequency division multiple access, 7
- FH, Frequency Hopping, 2
- FIFO, 30

**FlexRake**, 29
- Control, 31
- FlexRake receiver, 29
  - Correlator engine, 30
  - Stream buffer, 30

**Frequency Hopping**, 2

**Frequency hopping spread spectrum**, 3

**Handover**
- Soft and softer handover, 21

**Hybrid modulation**, 3

**Interference**
- Multiple access interference, 13
- Narrowband interference, 12
- Wideband interference, 12

**Interference rejection**, 7

**IS-95**, 2

**LPI, Low probability of interception**, 7

**Maximum Ratio Combiner**, 17, 28
- Multi path, 15
- Multiple access, 7
- Multiple access capability, 6

**Near far problem**, 13

**OVSF**
- Orthogonal variable spreading codes, 9

**Post buffer Rake receiver**, 32
- Channel compensator, 37

**Chip aligner**, 32
- Descrambler, 34
- Despreader, 34
- MRC, 37
- OVSF generator, 34
- PN generator, 33
- Symbol despread, 36

**Post Buffering Rake receiver**, 32
- Power control, 14
- Privacy, 7
- Processing gain, 2
- Protection against multipath interference, 6

**Rake**, 26
- Receiver, 17

**Rake receiver**
- Conventional, 28
- FlexRake, 29
- Post buffering Rake receiver, 32

**RF, Radio frequency**, 7

**SF**, 2

**SIMD**, 70

**Simulink**, 42

**Spreading codes**, 9
- Channelization code, 11
- Hadamard-Walsh codes, 11
- m-sequence, 9
- Scrambling codes, 10

**Synchronization**
- Acquisition phase, 23
- Tracking phase, 25

**SystemC**, 47

**TDMA, Time division multiple access**, 7

**TH, Time Hopping**, 2

**Time Hopping**, 2
- Time hopping spread spectrum, 3

**UMTS**, 2

**Variable Delay Buffer (VDB)**, 37

**VHDL**, 42

**VLIW**, 42, 47

**WCDMA**, 2

**Wire load models**, 57