

**Haverinen L. (2004) DSP Memory Management in a Third Generation High Performance Base Station.** University of Oulu, Department of Electrical and Information Engineering. Master's Thesis, 71 p.

## **ABSTRACT**

**Most of the tasks in a mobile cellular network base station are performed with programmable digital signal processors. Their memory spaces and management features are very limited. The buffering requirements in the base station can have large instantaneous variations during the simultaneous transmission of bursty data on multiple channels to multiple users. In particular the high bit-rates of the Wideband Code Division Multiple Access data transfer evolution High Speed Downlink Packet Access create very high demands for buffering.**

**The fragmentation of the buffer memory is a threat. It causes a gradual decrease in performance, which is critical in a long running process like the base station. The amount of fragmentation is different with different memory management methods.**

**In this work the features and applicability of different memory management methods for signal processors used in the base stations of third generation cellular networks have been studied. Software based memory management includes a high amount of conditional branches. The signal processor, which is optimized for highly parallel sequential computing, executes conditional branches very badly when compared to microcontrollers and general-purpose processors.**

**The memory management methods are first studied in theory and then experimentally. In the experiments two different memory management methods were analyzed. The memory managers were loaded with a synthetic workload program that simulates multi-user high bit-rate data transmissions in the base station. The performances of the memory managers were measured in terms of fragmentation, execution time and memory utilization.**

**The experiments confirmed the information gained from the theoretical studies that different memory management methods are usually optimized for a certain feature. The experiments showed that a simple method is fast to execute and works well with small and intermediate loads. When the load is increased the performance decreases. The second, more complex, measured method was found to require more computing, but to be capable of using the memory space assigned to it more effectively.**

**Key words: WCDMA, HSDPA, fragmentation, performance**

Haverinen L. (2004) DSP:n muistinhallinta kolmannen sukupolven suuren suorituskyvyn tukiasemissa. Oulun yliopisto, Sähkö- ja Tietotekniikan osasto. Diplomityö, 71 s.

## TIIVISTELMÄ

Pääosa matkaviestinverkon tukiaseman toiminnoista suoritetaan ohjelmoitavilla digitaalisilla signaaliprosessoreilla, joiden muistin koko ja – hallintaominaisuudet ovat rajallisia. Siirrettäessä puskemaista dataa samanaikaisesti usealla kanavalla useille käyttäjille aiheutuu puskurointitarpeille suuria hetkellisiä vaihteluita. Erityisesti Wideband Code-Division Multiple Accessin kehitysaskelen High Speed Downlink Packet Accessin suuret datanopeudet aiheuttavat erityisen suuria puskurointivaateita.

Puskurimuistin pirstoutumisen aiheuttama asteittainen suorituskyvyn lasku on tukiaseman kaltaisessa pitkään päällä olevassa järjestelmässä kriittistä. Muistinhallintamenetelmillä voidaan vaikuttaa pirstoutumisen määrään.

Tässä työssä on tutkittu erilaisten muistinhallintamenetelmien ominaisuuksia ja niiden soveltuvuutta kolmannen sukupolven matkapuhelinverkkojen tukiasemien signaaliprosessoreille. Ohjelmistopohjainen muistinhallinta käyttää runsaasti ehdollisia haarautumia, joita perättäiseen rinnakkaiseen laskentaan optimoitu signaaliprosessori suorittaa erittäin huonosti verrattuna mikrokontrollereihin ja yleisprosessoreihin.

Muistinhallintamenetelmiä tutkittiin ensin teoriapohjaisesti ja sitten kokeellisesti. Kokeissa testattiin kahteen eri menetelmään perustuvaa muistinhallintamenetelmää, joita kuormitettiin usean käyttäjän nopeaa tiedonsiirtoa vastaavalla keinokuormalla. Menetelmien suorituskykyä mitattiin pirstoutumisen, ajoajan ja muistin hyötykäytön suhteen.

Kokeet vahvistivat teorialuokituksissa saadun tiedon, että eri muistinhallintamenetelmät on yleensä optimoitu tiettyä ominaisuutta kohden. Kokeissa käytetyn yksinkertaisen menetelmän havaittiin olevan nopea suorittaa ja toimivan hyvin pienillä ja kohtuullisilla kuormilla, mutta suoriutuvan heikosti muistin hyötykäytöstä kuormituksen kasvaessa. Toisen käytetyn monimutkaisemman menetelmän havaittiin vaativan enemmän laskentaa, mutta kykenevän paremmin käyttämään hyväksi sille annetun muistimäärän.

Avainsanat: WCDMA, HSDPA, pirstoutuminen, suorituskyky

## TABLE OF CONTENTS

ABSTRACT .....	1
TIIVISTELMÄ .....	2
TABLE OF CONTENTS .....	3
PREFACE .....	5
ABBREVIATIONS .....	6
1. INTRODUCTION .....	8
1.1. Third generation cellular systems .....	8
1.2. Reaching for higher data rates .....	9
1.3. The role of memory management .....	10
1.4. Structure of the thesis .....	11
2. WIDEBAND CODE DIVISION MULTIPLE ACCESS CELLULAR NETWORK SYSTEM .....	12
2.1. UMTS air interface .....	13
2.2. UTRAN channel configuration .....	14
2.3. Base band digital signal processing aspects .....	15
2.4. High speed downlink packet access .....	16
2.4.1. Channel structure .....	18
2.4.2. MAC-hs .....	19
2.4.3. Flow control .....	21
2.4.4. Packet scheduler .....	23
2.4.5. Hybrid-ARQ .....	24
2.4.6. Adaptive modulation and coding .....	26
2.4.7. Architecture aspects .....	27
2.4.8. Performance .....	27
2.4.9. Requirements for memory management .....	28
3. BASE BAND DIGITAL SIGNAL PROCESSING ENVIRONMENT .....	31
3.1. Introduction to DSP processors .....	31
3.2. WCDMA Node B .....	32
3.3. Processor performance comparison .....	33
3.3.1. Texas Instruments TMS320C67x .....	33
3.3.2. Motorola G2 PowerPC 603e microprocessor .....	34
3.3.3. AMD Athlon .....	35
3.3.4. Summary .....	35
4. MEMORY MANAGEMENT .....	38
4.1. Dynamic memory management .....	38
4.2. Memory management problems .....	39
4.3. Memory fragmentation .....	40
4.4. Memory allocation policies .....	42
4.4.1. Common functionalities in all policies .....	42
4.4.2. Sequential fits .....	44
4.4.3. Segregated free lists .....	45
4.4.4. Buddy systems .....	46
4.5. Summary of memory management for DSP .....	46
5. MEMORY MANAGEMENT EXPERIMENTS FOR BASE BAND DSP .....	49
5.1. Description of the experiments .....	49
5.1.1. Descriptions of the measured memory managers .....	50

5.1.1.1. RTOS pools .....	50
5.1.1.2. Block Memory Manager .....	50
5.1.2. Synthetic workload program .....	51
5.2. Measurement metrics .....	52
5.2.1. External fragmentation .....	52
5.2.1.1. Linblad's method.....	52
5.2.1.2. Harrington's method.....	53
5.2.2. Internal fragmentation .....	54
5.2.3. Performance.....	55
5.3. Measurement methods.....	55
5.4. Test cases in the experiments .....	56
5.5. Results .....	57
5.5.1. Performance.....	57
5.5.2. External fragmentation .....	58
5.5.3. Internal fragmentation .....	59
5.5.4. Maximum User Amounts .....	60
5.6. Summary of the results .....	60
6. DISCUSSION AND FUTURE DEVELOPMENTS .....	62
6.1. Reaching the goals of the thesis .....	62
6.2. Enhancing the experiments .....	62
6.3. Ideas for developing DSP memory management .....	63
7. CONCLUSIONS .....	64
8. REFERENCES .....	66
9. LIST OF FIGURES.....	69
APPENDICES.....	70

## PREFACE

This thesis was written in the DSP Software unit of Nokia Technology Platforms division in Rusko, Oulu. The goal was to study DSP memory management in a high-speed base station.

I would like to thank Professor Olli Silvén at the University of Oulu for supervising this thesis and for his valuable help in acquiring a suitable subject for my thesis. A thanks goes also to the second supervisor, Professor Janne Heikkilä.

I would also like to thank my instructor, Jorma Taramaa, for his instructions and comments, which were very helpful during the whole writing process. Jouni Seppänen receives also my thanks for his help and advises.

I thank also my family, friends and all my co-workers for all their support and encouragement during my studies and the writing process of this thesis. Special thanks for my muse, Kirsi, for her wonderful musing.

Iso-Syöte, Pudasjärvi 2.4.2004

Lasse Haverinen

## ABBREVIATIONS

3G	Third Generation
3GPP	Third Generation Partnership Project
ACK	Positive acknowledgement, used in control signaling
ALU	Arithmetic Logic Unit
AMC	Adaptive Modulation and Coding
ARQ	Automatic Repeat Request
ASIC	Application-Specific Integrated Circuit
BMM	Block Memory Manager
BTB	Branch Target Buffer
CC	Chase Combining
CDMA	Code Division Multiple Access
CN	Core Network
CQI	Channel Quality Indicator
DL	Downlink
DPCH	Dedicated Physical Channel
DSP	Digital Signal Processor / Digital Signal Processing
FC	Flow Control
GRPS	General Packet Radio Service
GSM	Global System for Mobile Communications
H-ARQ/HARQ	Hybrid Automatic Repeat Request
HLR	Home Location Register
HSDPA	High Speed Downlink Packet Access
HS-DPCCH	High-Speed Dedicated Physical Control Channel
HS-DSCH	High-Speed Downlink Shared Channel
HS-PDSCH	High-Speed Physical Downlink Shared Channel
HS-SCCH	High-Speed Shared Control Channel
IR	Incremental Redundancy
MAC	Medium Access Control
ME	Mobile Equipment
MSC	Mobile Services Switching Centre
NACK	Negative acknowledgement, used in control signaling
PDU	Protocol Data Unit
PS	Packet Scheduler
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RLC	Radio Link Control
RNC	Radio Network Controller
RTOS	Real-Time Operating System
SAW	Stop And Wait
SDU	Service Data Unit
SGSN	Serving GPRS Support Node
TCP	Transmission Control Protocol
TDMA	Time-Division Multiple-Access
TFRI	Transport Format and Resource Indicator

TPC	Transmit Power Control
TTI	Transmission Time Interval
UE	User Equipment
UL	Uplink
VLIW	Very Long Instruction Word
UMTS	Universal Mobile Telecommunication System
USIM	UMTS Subscriber Identity Module
UTRAN	UMTS Terrestrial Radio Access Network
WCDMA	Wideband Code Division Multiple Access

# 1. INTRODUCTION

The third generation cellular systems offer much higher capabilities than any other previous cellular mobile communications systems. High capabilities come with high demands on hardware and software components. Demands are bandwidth and low latencies in the communication. New capabilities are required for new mobile applications like video streaming, file downloading, instant messaging, email and many other content rich services. These applications are especially demanding for the mobile last mile connections. The nature of the data traffic is highly bursty and high bandwidth usage with requirements for minimal latencies. Traditional voice traffic is also different than in the previous generation systems. Adaptive Multi-Rate speech codecs offering high coding densities generate variable bit rate traffic, which adds to the complexity.

## 1.1. Third generation cellular systems

The Third Generation Partnership Project (3GPP) specifications for the Universal Mobile Telecommunication System (UMTS) set the target bit rates to achieve at least 144kb/s and a goal of 384kb/s at rural outdoor environment. In suburban outdoors the bit rates get higher with a minimum of 384kb/s and goal of 512kb/s. Indoors the target is to have at least a 2Mb/s data rate. [1] These demands set high requirements for the base station signal processing elements, as the base station has to be able to serve multiple users. The numbers vary from dozens up to 32 and 64 users, depending on the manufacturer and model.

The base station has limited channel and code resources, which it can allocate to users. The number of simultaneous users and their connection speeds are directly dependant on the base station resources. These resources are set by the radio bandwidth given to the base station. The base station divides these resources not only in the time domain, but also in the code domain. In time domain it communicates with the users in short periods, 10ms in the Wideband Code Division Multiple Access (WCDMA) radio air-interface solution, and this way it can service multiple users at a given momentary time. Resource division in the code domain, known as Code Division Multiple Access (CDMA), means that the base station can multiplex several channels to users simultaneously at the same time. By allocating a different amount of these resources to the users it can serve multiple users with different bit rates according to their needs. Figure 1 shows an example of a base station communicating with multiple users simultaneously with different data rates.

Here the base station has divided its resources to different users completely. If a new user would require a connection these resources would have to be taken from someone else. For example dropping the data rate of the 512kb/s connection could do this.

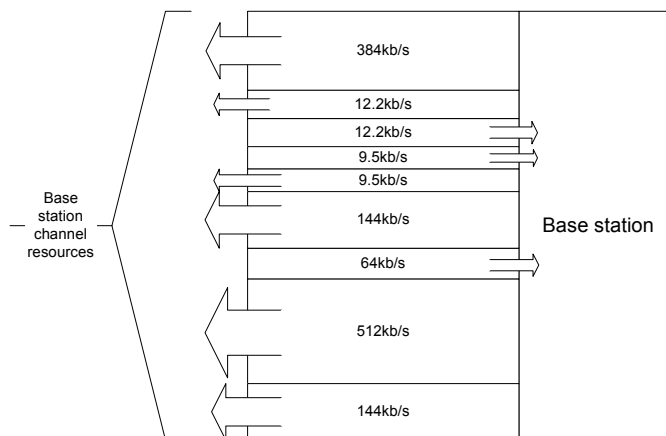


Figure 1. Example of base station code resource division.

The figure shows the situation for a given moment of time and does not portray the time-sharing capabilities or the changing resource allocations. The figure shows the idea that there can be a great diversity in the data rates of each user. It is this multichannel and code diversity with a great variety of possible data rates that make the task of the base station signal processing hard. Each connection needs to be buffered for flow control, possible automatic-repeat request protocols and other such purposes. The faster the connection the more memory it needs for buffering and the more carefully the memory has to be managed. If the memory management does not perform well it can bring to a halt or degrade the performance of the whole base station. Or it can just waste memory resources, which have to be compensated by adding more memory. Whatever the effect, it will be a severe disadvantage in the highly competed mobile network business. Memory management is a small, but critical task in the third generation high performance base stations.

## 1.2. Reaching for higher data rates

The 2Mb/s bit rates targeted in the first UMTS specifications are not even near the maximum bit rates that the WCDMA system can offer. UMTS and the WCDMA air interface are specified by 3GPP in its so-called Release 4 specifications. Previous releases are for GSM systems. Release 4 requirements state that it is desirable that the definition of the UMTS Radio Access Network (UTRAN) should allow evolution to higher bit rates [1]. This evolution is specified in the later Release 5. Among other features the Rel5 introduces the WCDMA air interface improvement called High-Speed Downlink Packet Data (HSDPA).

HSDPA is a feature based on a downlink shared channel, data only, that allows data rates of up to 10 Mb/s. It is designed to support services that require instantaneous high rates in the downlink and lower rates uplink (also called "reverse link"). This feature also allows to decrease the level of retransmissions (at the Radio Link and hence higher layers), in turn allowing the reduction of delivery time. Examples of end-user services using HSDPA are Internet browsing and video on demand. The nature of HSDPA makes it very bursty on the application level and physical radio link level. [2]

### 1.3. The role of memory management

HSDPA increases the importance and the role of the base station in the communication chain. The base station has to be able to serve multiple users who are using megabyte class bit rates simultaneously. This requires much more memory and signal processing capabilities than in the previous Release 4, which itself is also a demanding specification. Because of these high demands of the task in terms of signal processing power, the only cost-effective and reasonable solution for implementation is a programmable Digital Signal Processor (DSP).

Another key factor is the memory and how it is managed, as already mentioned before. Signal processing tasks that these DSPs are designed for are tight loops that are typically highly predictive and repetitive with many parallel computation operations. Memory management, on the other hand, involves lots of comparisons and conditional decisions leading to unpredictable branches in the execution sequence, the complete opposite of a typical signal-processing task.

In Figure 2 the scenario a) at the left shows the typical task sequence for which the DSP is designed and optimized. Scenario b) presents a typical situation of a memory management code where execution is heavily unpredictable, has high dependencies and cannot be effectively scheduled. This kind of situation effectively negates the innate capabilities of a DSP in multiple parallel multiply-accumulate and arithmetic calculation operations, if the system has to wait for the results of branches.

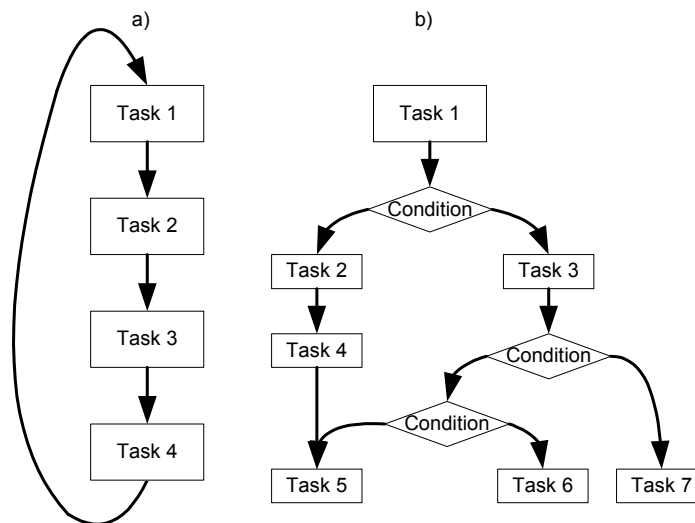


Figure 2. Differences of typical signal processing and memory management sequences.

There are many memory management techniques that have each been designed to optimize various factors. Some have low processing overhead, some exhibit low fragmentation tendency and some are designed for very large memory spaces. Many of these techniques require intensive computation with many branches, which are usually handled with a dedicated hardware memory management unit or branch prediction units. In microcontrollers and general-purpose processors this kind of hardware support is common. In DSPs hardware support for memory management is

rare. The challenge is to find such method for DSP memory management that it is light enough and still offers good enough performance.

No previous public researches about memory management with DSP have been found before the writing of this thesis. Therefore it can be safely said that publications of memory management with DSPs are very rare. In this thesis DSP memory management will be studied and the performance of two memory managers measured. The goal of the work is to find methods for measuring the performance of memory management, measure the performance of memory management methods with a DSP and to perform an evaluation of these methods from the viewpoint of DSP in a HSDPA base station. The performances of two memory managers with different operating principles are measured with a cycle-accurate DSP simulator. Memory managers are loaded by generating synthetic memory requests with software that has been created for the purpose. The memory requests made by the software try to simulate the memory activity encountered in a multi-user HSDPA environment. The performance is measured with selected metrics for four quantities: external fragmentation, internal fragmentation, memory utilization and processing cycles used for the memory management. Then the results of the measurements are reflected against the previous findings on memory management for DSPs.

#### **1.4. Structure of the thesis**

The structure of the thesis is as follows: chapter 2 presents the functional background and principles of UMTS and HSDPA and is concluded by a summary of the requirements presented to the DSP. Chapter 3 introduces the environment for the base band digital signal processing. It begins with an introduction to DSP processors, then presents the general structure of a HSDPA base station and ends with a comparison of different processor architectures. Architectures are compared to gain a view of typical DSP architectural drawbacks when processing heavily branched control-code, which the memory management is all about. Chapter 4 presents the theory and concepts of memory management methods and evaluates the costs and benefits of these methods for the DSP. Chapter 5 presents the methods and metrics for the memory management experiments. It introduces the test cases and the environment in which the testing takes place. The chapter ends with an examination of the results achieved with the experiments. Chapter 6 has ideas for future developments for memory management with DSP and for the testing procedures. Chapter 7 ends the thesis with overall conclusions and summarizes the results achieved and their significance.

## 2. WIDEBAND CODE DIVISION MULTIPLE ACCESS CELLULAR NETWORK SYSTEM

This chapter presents the structure and main principles of the Wideband Code Division Multiple Access (WCDMA) cellular network system, also known as the Universal Mobile Telecommunication System (UMTS). UMTS provides the high bit rates that are required to provide the multimedia services so highly anticipated and marketed with 3G systems. The target capability has been defined at 384kb/s for large areas and 2Mb/s for smaller local areas. The general architecture of the UMTS network is illustrated in Figure 3.

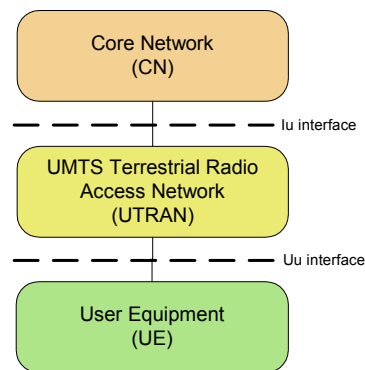


Figure 3. Conceptual UMTS network architecture.

The basic building blocks of the UMTS network architecture are the Core Network (CN), UMTS Terrestrial Radio Access Network (UTRAN) and the User Equipment (UE). These elements are connected to each other via open the Uu and Iu interfaces specified in the 3GPP standards to enable the communications of products from different manufacturers.

The Core Network is a combination of exchanges and transmission equipment used to connect the incoming and outgoing traffic to the subscribers and external networks. UTRAN provides the access network for the UE to reach the Core Network. Most of this thesis concentrates to the UTRAN. User Equipment gives the user the means to reach external networks using the services provided by the UTRAN and CN. [1]

A more detailed view of the network elements in a UMTS network is displayed in Figure 4. The Home Location Register (HLR) stores the master copy of the user's service profile, which holds information about allowed services, roaming areas, status of call forwarding and such. The Mobile Switching Center (MSC) is a CN element, which performs the switching functions in its area of operation. VLR holds information on the visiting user's service profile and some location information in the serving system area.

The Serving GPRS Support Node (SGSN) holds the functions of MSC/VLR, but for the packet switched services. Gateway MSC (GMSC) is the location where the UMTS network is connected to a external circuit switched (CS) network. All

incoming/outgoing CS connections go through GMSC. Gateway GPRS Support Node (GGSN) does the same thing for packet switched traffic.

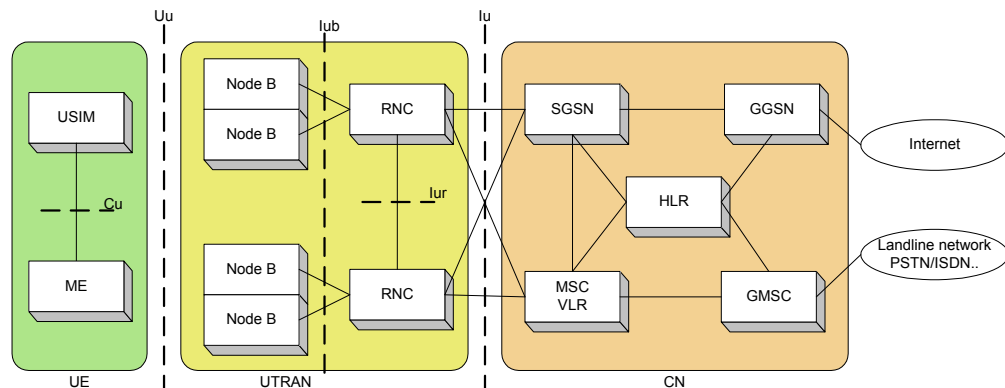


Figure 4. Network elements of a UMTS network.

The UTRAN represents the Access Network (AN) infrastructure for the UMTS network. UTRAN consists of one Radio Network Controller (RNC) and one or more Node Bs. RNC controls the radio resources (Node Bs) assigned to it. RNCs use the Iur interface for communicating between each other. Node B is the UMTS equivalent of the traditional base station, which handles traffic within one particular cell. Node B and RNC communicate through the Iub interface and the radio interface from the Node B to the mobile station is called Uu.

UE describes the equipment used by the subscriber and consists of the UMTS Subscriber Identity Module (USIM) and the mobile terminal itself. A terminal can be either a dual mode mobile that operates in both GSM and UMTS networks or a dedicated UMTS mobile. USIM holds all the subscriber information required by the terminal to operate. For communication between the USIM and mobile terminal the Cu interface has been defined.

## 2.1. UMTS air interface

The UMTS WCDMA solution operates with a 5MHz bandwidth and uses Direct Sequence Code Division for its multiple access method (DS-SS). Direct sequence is a form of spread spectrum technique. CDMA enables multiple users to use the same frequency band simultaneously. Users are assigned a code/codes and those codes are used to separate the users. Frequency division is used for separating the uplink and downlink transmissions. The downlink frequencies are in the range of 2110 – 2170 MHz and uplink between 1920 – 1980 MHz.

In direct sequence spread spectrum the data signal is spread over a much wider band than the bandwidth of the actual information. The transmitter spreads the signal by convoluting the data with a high bit rate spreader code. This produces a wideband signal and is known as direct sequence spreading. The receiver of this signal then despreads the signal by multiplying it with the exactly same spreader code. This results the original signal plus some possible high frequency noise components from the radio transmission that are easy to filter. Now the signal is ready for channel decoding and further processing, usually done with DSP software.

## 2.2. UTRAN channel configuration

Different channels are used for transmitting all the communications between the user equipment and the UTRAN. These communications include everything from speech and data transmissions to signaling functions that the UMTS needs for its services. A channel refers to bandwidth and its controlling functions allocated to the user by the RAN. UTRAN uses a three-channel organization, which consists of logical, transport and physical channels. The location of each channel in the UTRAN concept is shown in Figure 5.

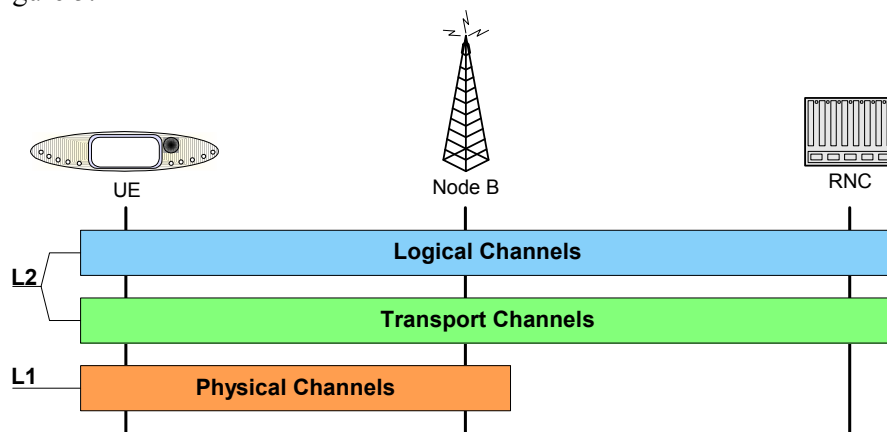


Figure 5. UTRAN Channel types and their locations.

Logical channels describe the type of the information, which is transmitted, and these channels are offered to the higher layers. Transport channels are channels offered by the physical layer (L1) to the logical channels in layer 2 for data transport purposes. Physical channels are the actual transmission medium, the radio system, through which the information is transferred. Information to these channels is multiplexed from the logical channels. [4]

Transport channel characteristics are defined by its transport format, which specifies the physical layer processing to be applied to the transport channel. Processing includes such tasks as channel coding, interleaving and any specific rate matching if needed. For the transmission of data in the physical channels a Transport Block (TB) is defined as the data accepted by the physical layer for encoding. A transport block has to be created every 10ms, or multiple of 10 ms, according to the radio frame timing specified in the 3GPP technical specifications. The basic requirements for signal processing come from these physical layer specifications. The physical layer is the lowest layer, Layer 1, in the OSI Reference Model. [5]

Channels are mapped from the logical to transport to physical channels in both down- and uplink directions. All three levels of channels and their downlink channel mapping can be seen in Figure 6.

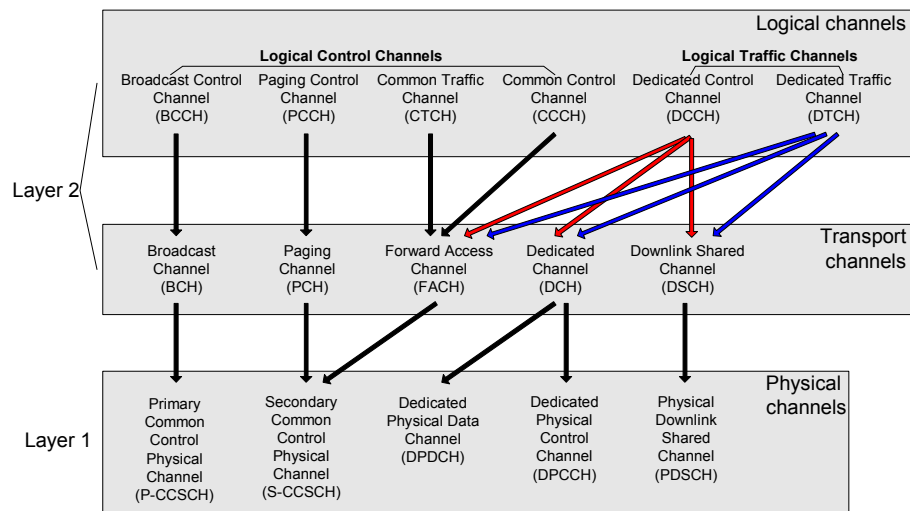


Figure 6. Logical to transport to physical channel mapping in downlink direction.

In the uplink direction the required amount of logical channels is smaller. Only three logical channels are required. The mappings between logical to transport and transport to physical channels is seen in Figure 7.

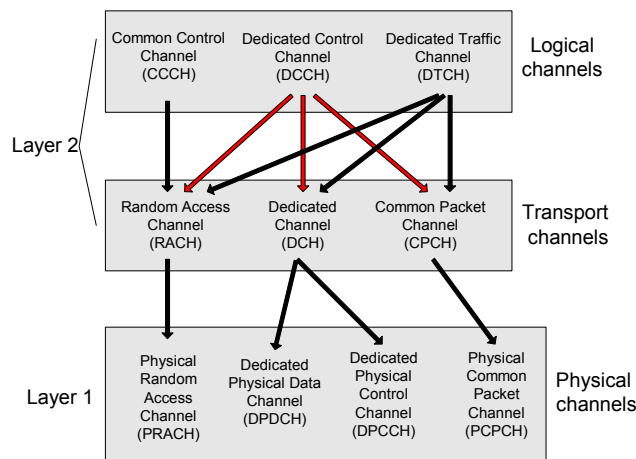


Figure 7. Logical to transport to physical channel mapping in uplink direction.

The common channels are used to serve multiple users at the same time. The dedicated channels are user dedicated point-to-point communications between one UE and a Node B.

### 2.3. Base band digital signal processing aspects

Encoding and decoding of the physical channels in the layer 1 is performed in the Node B by DSP software and the processing unit is known as Codec DSP. The main tasks of the Codec DSP are:

- Rate matching / dematching,
- Interleaving / deinterleaving,

- Convolutional encoding,
- CRC calculation / checking,
- Turbo encoding,
- Transport channel demultiplexing, and
- Other controlling and measurement functions.

Usually DSP loads are targeted above 90% of their gross computing power to gain maximum performance per dollar. In any computation there is some control code among the actual performing code. The relationship of the amounts of control vs. performing code decides the final net computing power achievable that is actually available to advance the task at hand. Typically a modern cellular base station is designed to have some extra processing power available. This is to allow possible future enhancements with new features and to increase the number of maximum users per processing unit. The amount of users served depends on the processing speed and memory of the DSP and user bit rate required from it. Estimations are that with 384 kbps user bit rates the amount of simultaneous users is eight times less when compared with an 8-16 kbps bit rate. [5]

Program code for the required functions should fit into the program memory to avoid fetching instructions from external memory, which usually quickly becomes the bottleneck for the whole system. Program memory requirements are estimated to be more than 350kB, which is more than most of the currently commercially available DSPs can provide. Data memory requirements with high data rates are such that external memory is required to satisfy the needs. An estimation is that decoding one 384 kbps transmission would require as much as 250kB of data memory. The actual amount varies from 0 to 250 because of the bursty traffic. If one DSP could serve multiple users with high data rates the absolute worst-case requirement for the amount of external memory increases quickly. For example with four users the worst-case required memory would be over 1MB. It is clear that with multiple simultaneous channels operating with a variety of a data rates, very careful memory management is needed.

Traffic and control signaling channels are not synchronized to each other and the nature of the traffic is very bursty. When combined with the fact that there will be multiple users using multiple simultaneous channels, the result is that the memory management method used has to provide real-time response times. The high requirements of high bit rate transmissions are the reason for the fact that initial public WCDMA networks will support only lower rates at first. [5]

#### **2.4. High speed downlink packet access**

Growing needs for high capacity data transmissions beyond the first third-generation requirements are the main reasons for the development of the WCDMA air interface enhancement called High Speed Downlink Packet Access (HSDPA). HSDPA is defined in 3GPP Rel-5 specifications. This section introduces the concept and main features of the HSDPA.

HSDPA will improve WCDMA downlink data transfer services with smaller delays and data rates with a theoretical maximum of 14.4Mb/s, and practical peak rates up to 8-10Mb/s. [5] To achieve this HSDPA uses fast link adaptation, high order modulation, a new high speed MAC layer containing fast hybrid automatic repeat request (HARQ) and the fast scheduling functionality shown in the features

diagram of Figure 8. The Adaptive Modulation Coding block in the figure means combined link adaptation and modulation coding. HSDPA can be seen as an evolution of WCDMA as EDGE was an evolution of GSM towards the 2.5/3G. [6]

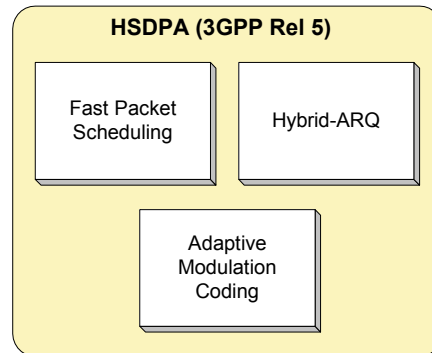


Figure 8. HSDPA Features.

High order modulation (e.g. 16 Quadrature Amplitude Modulation (QAM)) can be used to provide high peak data rates because of the increased spectral efficiency in terms of bit/s/Hz when compared to Quadrature Phase Shift Keying (QPSK). High order modulation is enabled by the good signal-to-interference ratios achieved by directing large part of downlink power to a single user at good channel conditions. Fast link adaptation should be used with high order modulation if the modulation and coding are to be adapted to the instantaneous channel conditions. With fast link adaptation high order modulation and high-rate coding can be assigned to users in good channel conditions and likewise use low-rate coding and strong QPSK modulation for users in the less favorable channel conditions. [7]

With basic automatic repeat request protocols the receiver detects any erroneous packets and requests a retransmission of that packet. With HARQ the receiver combines information from both the original transmission and from the required retransmissions before any decoding attempts. Therefore using HARQ together with fast link adaptation is useful. Link adaptation provides an initial estimate of the required redundancy to avoid an excessive number of retransmissions and HARQ compensates the errors in the channel quality estimates used by the link adaptation. [7]

Fast scheduling is the key for any packet data system as it determines which user is in turn to transmit in the given Transmission Time Interval (TTI) and this way largely determines the whole behavior of the system. By using different scheduling algorithms the system operators can customize the system to their needs. Scheduling the radio resources to the users in the best channel conditions, while taking Quality of Service (QoS) into account, results in maximum cell throughput. [7]

HSDPA features present high demands for DSP hardware and software. Processing power requirements are such that satisfying them with current hardware probably requires one or two dedicated DSP chips in the base station signal processing boards. In addition to the processing power, another key issue is the amount memory and how it is managed. With data rates up to 10Mbit/s there has to be minimal delays between the transmitted packets and especially in delay times of retransmissions. Therefore buffered data for several TTIs per user in the Node B itself must be readily available, since the delay time of fetching the data from the RNC or from the Core Network is unacceptable. The nature of the data transmissions

is that they are bursty and in the form of packets. Every data packet has to be buffered. Packets are buffered first in the flow control and then in the HARQ for the possible retransmission. These buffering operations and the associated memory allocations have to be fast, effective and simple.

### 2.4.1. Channel structure

HSDPA introduces quite a few new channels. These new channels that are related with HSDPA are displayed in Figure 9. Node B controls the HSDPA channels (High Speed – Dedicated Physical Control Channel (HS-DPCCH), High Speed – Shared Control Channel (HS-SCCH) and High Speed – Downlink Shared Channel (HS-DSCH)) with the MAC-hs protocol. [8]

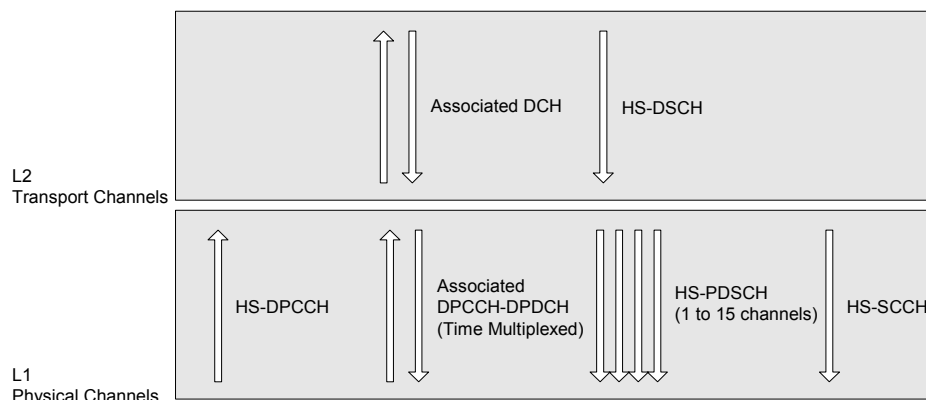


Figure 9. HSDPA Related channels (for one user).

The uplink HS-DPCCH and downlink HS-SCCH are used for physical layer HS-DSCH signaling. HS-DPCCH is used by the UE to transmit channel quality information and ACK/NACK signals to the HARQ entities in the Node B. HS-SCCH is used by the Node B for signaling link and channel adaptation information to the UE. This information consists of channelization codes for despreading, used modulation & coding and HARQ information, all of which are necessary to the UE to receive the incoming data from HS-DSCH. [9]

When an UE enters the HSDPA mode it is assigned an Associated DCH channel and a HS-DSCH is set up on the L2 layer for the user on the DCH channel. For transmission the data is mapped onto the L1 HS-PDSCH (1 to 15 physical code channels, depends on the required data rate and modulation). HS-DSCH may carry multiple data streams for one UE, each with different priorities. The data for each priority class is determined by its category. These categories are streaming, interactive and background. The amount of HS-DSCHs is limited by the fact that for each HS-DSCH there has to be one HS-SCCH and one UE can listen only to a maximum of 4 HS-SCCH channels at a time. [8]

HS-DSCH code resources are shared primarily in the time-domain (TDMA) among several users. The allocation of resources in HS-DSCH is done on a TTI basis. TTI time is chosen to minimize delays, allow fine granularity in the scheduling and to be able to follow channel quality variations and still enable reasonable data transmission sizes when operating with limited code resources. To meet these needs

the HS-DSCH TTI is chosen to be 2ms. The second method of sharing the code resources is code-multiplexing (CDMA) where data for more than one user is transmitted on the HS-DSCH at the same time using a set of unrelated codes. In the target scenario the Node B has 15 codes, which it can divide between the UE's. For example three users could be served concurrently with five codes assigned to each of them. This situation can arise if the data of the first user doesn't fill the entire channel capacity or if the UE is not capable of supporting the entire channelization code resource range. Therefore it is beneficial to be able to exploit the unused capacity with code multiplexing to achieve maximum system throughput. [7]

For DSP memory management this kind of mix of CDMA and TDMA methods makes the task tricky. If three users are served within one TTI, the software and memory manager must be able to perform these multiple memory accesses per user in a much smaller time window that a TTI is. This is because there will be numerous other tasks for the DSP to perform during one TTI. The main components of these activities are presented in the following sections. For transmission the data has to be assembled from flow control buffers to actual transmitted packets. These packets have to be buffered in the ARQ buffer. Even in this kind of simple situation there will be numerous memory accesses whose total processing time cannot be very high.

#### 2.4.2. MAC-hs

HSDPA increments the existing MAC layer in the WCDMA with a new functional entity, the MAC-hs. The MAC entities in the UTRAN are MAC-d, MAC-c/sh and MAC-hs as seen in Figure 10. which displays their corresponding channels. They control accesses to specific transport channels.

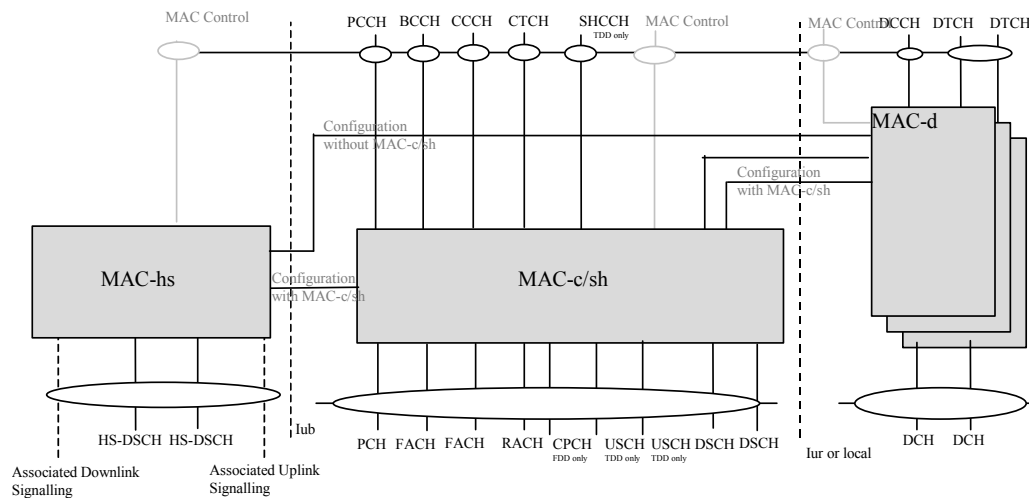


Figure 10. UTRAN MAC architecture.

MAC-d controls the dedicated transport channels, MAC-c/sh controls the common and shared channels and MAC-hs controls the HS-DSCH by mapping logical channels from MAC-d to HS-DSCH. All MAC entities are located in the RNC except for the MAC-hs, which is located in the Node B.

The Protocol Data Unit (PDU) provided by MAC-d has a total length of 340 bytes. The structure of the PDU is displayed in Figure 11. The first field is the Target Channel Type Field, which indicates the information carried by the PDU. UE-Id Type is used for correct decoding of the following UE-Id, which identifies the recipient UE on the common transport channels. The C/T field provides identification of the logical channel instance when multiple logical channels are carried on the same transport channel [8].

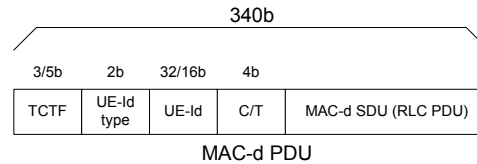


Figure 11. MAC-d PDU structure.

Figure 12. shows the different functional entities inside the MAC-hs: Flow control, HARQ control, Link adaptation and Packet scheduling. Each of these are studied more carefully in their own chapters later. [8] MAC-hs take in MAC-d PDUs (Service Data Units (SDUs) from the MAC-hs point of view) for processing and produces MAC-hs PDUs. One MAC-hs PDU, also known as Transport Block (TB), holds many MAC-d PDUs. Number of MAC-d PDUs in the MAC-hs PDU depends on the size of the TB. And the size of the TB depends on the data rate of HSDPA transmission.

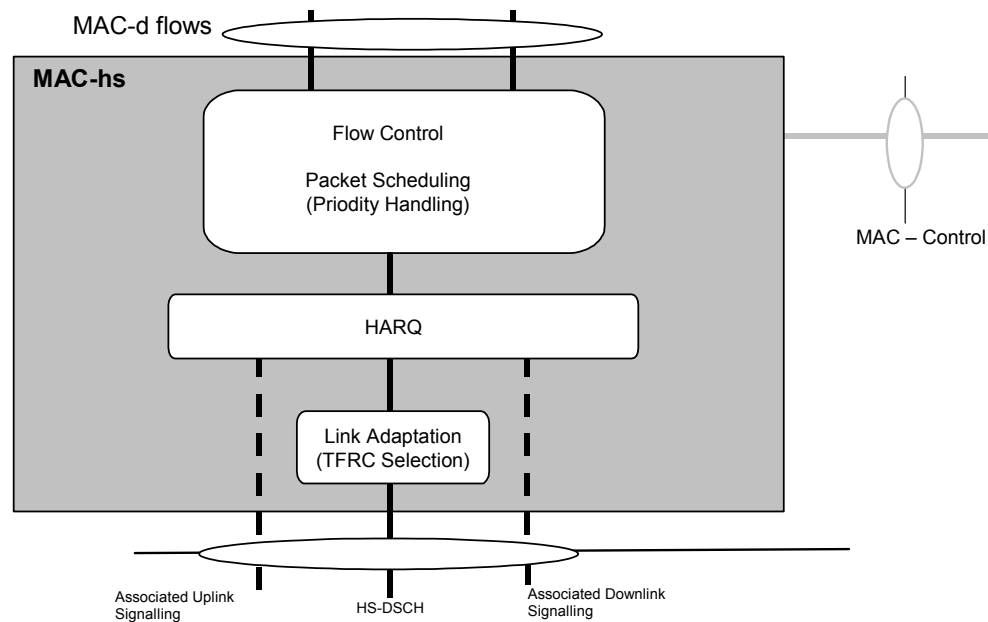


Figure 12. MAC-hs detailed architecture.

A Transport Block contains only MAC-d PDUs from the same priority category and user. Transport Block is the data packet that the UE receives in a HSDPA transmission. The detailed structure of the MAC-hs PDU is seen in Figure 13. The first field is the one bit Version Flag and in the 3GPP Rel5 the value will always be

set to 0 and 1 is reserved for a future use. Queue Id identifies the reordering queue for the receiver. TSN or Transmission Sequence Number identifies the sequence on the HS-DSCH and is used for in-order data delivery to the higher layers in the OSI model. Size Index Identifier, SID, gives the size for consecutive MAC-d PDUs in the same set. N, Number of MAC-d PDUs gives the number of MAC-d PDUs in the size set. The last field is the Flag (F), which tells whether more SID-fields are present in the header or not. If the F is set to 1 then MAC-d PDU follows the field. The limit of MAC-d PDUs in the MAC-hs PDU is 70. The end of MAC-hs PDU is padded with zeroes if the combined header and payload data do not correspond to any of the MAC-hs PDU sizes defined by the 3GPP Medium Access Control Protocol specification. [8]

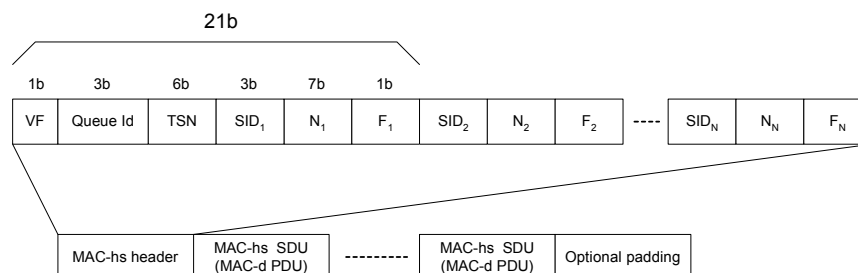


Figure 13. MAC-hs PDU.

The motivation for locating the MAC-hs in Node B is that this enables fast physical layer (L1) retransmissions using HARQ and this results in less delay jitter and better conditions for data services like TCP. Another advantage is that it enables fast and accurate air interface measurements, which are required by other HSDPA features for efficient operation.

### 2.4.3. Flow control

Flow Control (FC) is an essential component securing for effective data transfer services. The task of FC in MAC-hs is to maintain buffers of MAC-d PDUs for all currently active HSDPA users. The FC in the MAC-hs is a companion to the FC in MAC-d, or MAC-c/sh if configured so. Together they provide a controlled flow of data between the two MAC entities. HS-DSCH Frame Protocol (FP) handles the data transport over Iub/Iur –interfaces from RNC to Node B. An illustration of this is seen in Figure 14. HS-DSCH FP typically transports multiple MAC-d PDUs of same size and priority within the same data frame. MAC-d PDUs are then buffered in the MAC-hs in priority queues, or buffers, per user and per priority level.

The transmission interval on Iub-interface is 10ms and round trip time approximately 20ms. This means that in Node B there has to be buffered data for at least 20ms (10 TTIs) per each user in priority queues. The amount of buffered data for one TTI depends on the user bit rate, because one Transport Block will be sent to user in a TTI. Maximum TB size is 28800 bits and this sets the required maximum amount of buffered data per TTI. In addition each user has multiple priority queues so the amount of buffers needed increases, too. [10]

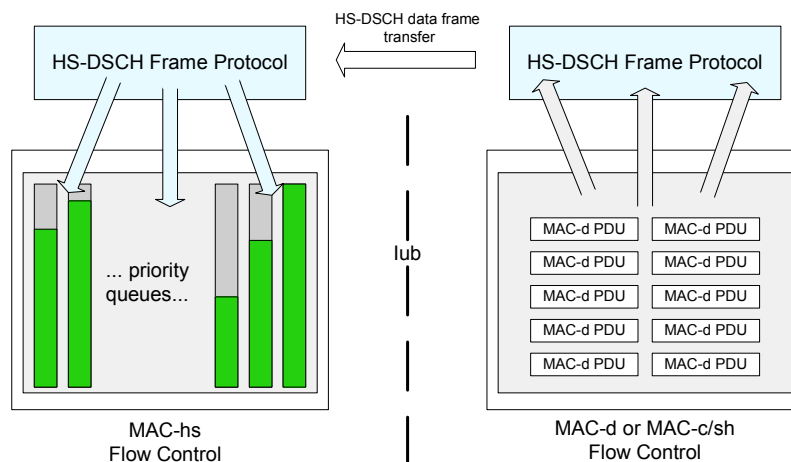


Figure 14. Flow control principle.

Flow control monitors the status of the priority queues and when the fill level of the buffers sink too low it requests replenishment transfer from the RNC. This is called capacity allocation. There are different strategies for capacity allocation. For example capacity can be allocated for a user with the least amount of data in the buffer, or for the user whose buffer is emptied at the fastest rate.

In Table 1 there are listed estimations of the maximum memory requirements of the FC with 28800-bit TB size and with different numbers of variables. This TB size corresponds to the HSDPA data rate. The smallest and largest values in the table have a difference of 12375 kilobytes. For example the TMS320C6000 DSP series from Texas Instruments can address up to 256 megabytes of external memory, but only with double word mode. With byte addressing the maximum amount drops to 32 megabytes. With this background it can be said that the several megabyte requirement of the flow control buffers is quite demanding.

Table 1. Estimations of Flow Control memory requirements

No. of priority queues / user	2	2	6	6	8	8
No. of users	16	32	16	32	16	32
Buffer length (TTIs)	10	10	10	10	15	15
Required memory in kilobytes	1125	2250	3375	6750	6750	13500

Data handled by the Flow Control is especially demanding for the memory space and does not have any hard real-time requirements built-in by nature. The large amounts of the data will however, lead to large amounts of memory operations in short time frames. This forces the memory manager to have small allocation and deallocation latency, because Flow Control will request these operations frequently. The data incoming from the HS-DSCH frame protocol are MAC-d PDUs, which have a size of 340 bits.

This means that allocations made by flow control have a fixed size. As will later be seen, this can be effectively used as an advantage to minimize fragmentation. The theoretical amount of allocations for situation seen on the second column in Table 1 would be  $18432000/340 = 54212$  allocations. This example would only be true if the

priority queues of all 32 users were empty and suddenly simultaneously filled. However, this example shows that there will be a very high amount of memory operations with the Flow Control and because of this the operations have to be done very fast, yet effectively to minimize fragmentation and the processing time lost with managing the memory.

#### *2.4.4. Packet scheduler*

The Packet Scheduler (PS) is located at Node B and it makes fast scheduling optimized to the current channel condition possible. The task of the PS is to decide which user, or users if code resources allow it, is in turn to be served on the HS-DSCH and controls its operation with the HS-SCCH. Each cell employs one PS, which schedules all UEs within the cell. PS is also involved with the automatic retransmission decisions indirectly. PS selects the UE in turn and informs the HARQ of this decision, which then in turn makes the decision whether to send a retransmission or a new transmission. [11]

A TTI time of 2ms forces the PS algorithm to make its decisions very quickly. There are several fast scheduling algorithms available to be used with the PS in HSDPA implementation. Each of them has different qualities for fairness among the users and between cell throughputs. Some suggested algorithms are:

- Fair Resource, where all UEs get an equal amount of power, code and time.
- Fair Throughput, where all UEs get the same data throughput.
- Maximum Throughput, where the UE in the best channel conditions gets all the available resources.
- Proportional Fair Resource, where the UEs in the highest relative instantaneous channel quality gets the available resources.

The Proportional Fair Resource (P-FR) scheduling algorithm offers the most tempting solution to the PS since it provides fairness among UEs, high cell throughput and still guarantees minimum QoS. The P-FR algorithm schedules to the users who are on the top of the radio link fades. This results in optimal cell capacity. Fast channel quality information is required to make P-FR scheduling possible and therefore the scheduler needs access to the link quality information for all UEs in the queue to be able to operate at full capacity. An example of this principle can be seen in Figure 15, which displays the good scheduling possibilities and as an example one bad scheduling selection. [11]

The figure shows that the operation idea of radio link makes the traffic highly bursty. The reason for this is that one or few users receive all the available radio resources and therefore each user receives high bit-rate data periodically. Accommodating this bursty traffic is hard for the memory management handling the buffers.

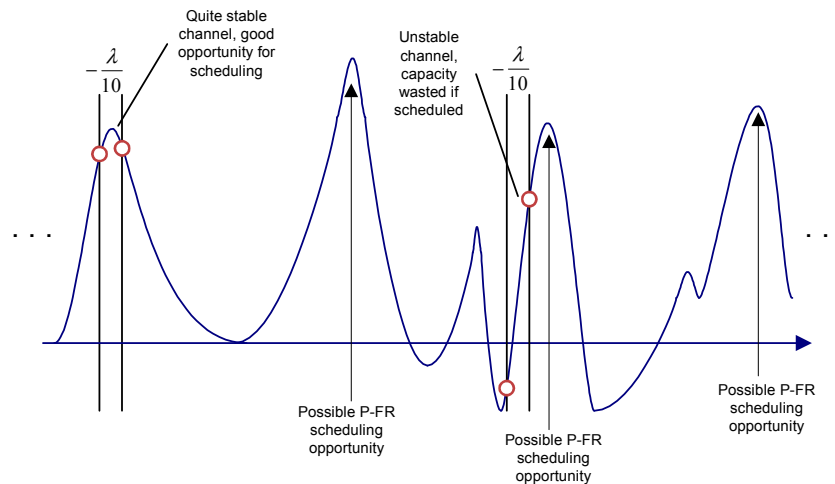


Figure 15. Proportional Fair Resource and the top of the fades.

P-FR and the top of the fades strategy is also known as fast selection multi-user diversity transmission. Depending on the UE situation this strategy can provide cell throughput gain of up to 56%, when compared to blind scheduling where no a priori knowledge of the radio channel is used. [11]

#### 2.4.5. Hybrid-ARQ

Hybrid Automatic Repeat Request (HARQ) in HSDPA uses N-Channel Stop-And-Wait (SAW) protocol. In SAW the transmitter keeps sending the data until the receiver successfully receives it. N-Channel SAW doesn't stop to wait for acknowledgements, but transmits another one. To avoid stalls when waiting for acknowledgements (ACK) from a UE, a maximum of eight SAW-ARQ processes can be run for a single UE, so that they operate in different TTIs. This means that a new HARQ process is started for each transmitted TB and ended when an ACK is received from UE signaling that all data in a TTI has been successfully received. If the data has been decoded with errors then a retransmission is scheduled within the next few TTIs. The retransmitted data is then combined with the original transmission before decoding and greatly increases the probability of successful decoding. [7]

The HARQ is located in the MAC-hs in the Node B and the RNC is not involved in any way. This reduces the retransmission delays by several orders of magnitude, compared to the conventional solution where ARQ is handled by the RNC. In the RNC ARQ the combining of different transmissions is not possible, since the fast data rates of HSDPA and long retransmission delays require buffer sizes that would be too large for the UE. The ARQ Combining scheme is based on Incremental Redundancy. HARQ reduces the average number of transmissions, but the tradeoff is that each transmission carries redundant information. [11]

The HARQ function is made up of the physical layer (L1) the HS-DSCH encoding part and of the MAC layer (L2) control part. The control part is implemented in the MAC-hs Packet Scheduler. Control decides when to retransmit the same packet again based on the ACK/NACK –reports from the UE sent in the HS-DPCCH. In the

MAC-hs there is one HARQ entity per UE and one entity handles the HARQ functionality for one user. One HARQ entity can have N amount of HARQ processes for the same UE, each having round trip time (RTT) for retransmissions  $N \cdot TTI$  ( $N \cdot 2ms$ ). The maximum value for N is 8, but in practice the value is less.

Two combining schemes are supported in the HSDPA concept: Incremental Redundancy (IR) and Chase Combining (CC), a particular form of IR. The principle of CC is shown in Figure 16. The idea is to transmit an identical version of a packet for which the UE has signaled NACK and then the decoder in the UE has to combine the received copies and weight them by their corresponding signal to noise ratios before combining. [7]

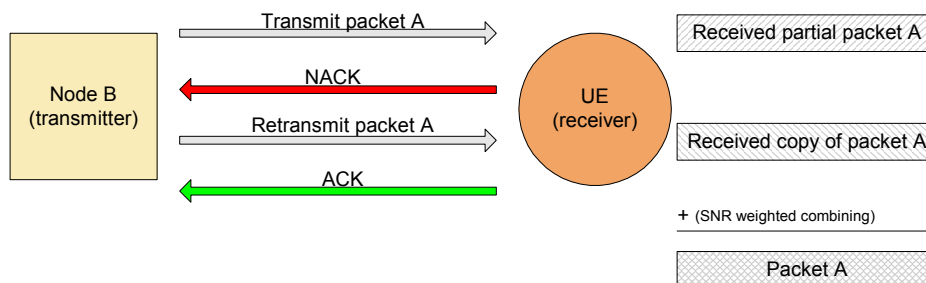


Figure 16. Principle of Chase Combining.

In the IR scheme the transmitter incrementally adds redundant coding information to the transmitted packet if the receiver fails to decode the packet on the first attempt. Information bits are encoded with a low rate code, and then the information and selected number of parity bits are transmitted. If a retransmission is not successful then the transmitter sends additional parity bits and the receiver combines the new bits and those previously received. The selection of the bits from the mother code is called puncturing. The principle of IR with 1/5 mother code is shown in Figure 17, which displays with different colors the bits in the transmitter, the transmission order and the bits that are received.

Each retransmission therefore produces a codeword of a stronger code. At coding rates of  $\frac{3}{4}$  the IR scheme is close to optimum ( $\frac{1}{3}$  is the base encoder rate) after the second transmission. For a code rate of  $\frac{1}{2}$  or less the IR doesn't provide any significant advantage over CC, because that nearly all of the code information has been sent in the first transmission.

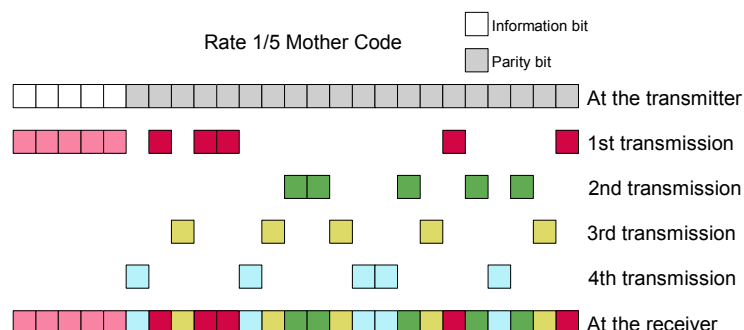


Figure 17. Principle of Incremental Redundancy.

When UE memory requirements are compared, the CC has a clear advantage, since IR requires relatively high amounts of memory. [11]

HARQ is a demanding part of the HSDPA. Each HARQ process has to be buffered in the memory to wait for ACK/NACKs from the UE. The amount of memory required to buffer a single HARQ process depends on the TB size and the memory management used. If the space for HARQ is statically allocated then the required amount is equivalent to the maximum TB size (eg. 28800b). In this case the TBs are always placed in the same place in the memory. There has to be space for the possible worst-case scenario (e.g. Max TB size), even if no single UE supports the maximum TB size. This applies if the system is intended to be operational longer than the initial HSDPA phase-in when the supported data rates are circa 1/3 of the maximum.

In the case of dynamic memory management the required memory for a HARQ process depends on the size of the TB. The memory space for the buffered TB is allocated by the size of the TB. When the whole HARQ functionality is viewed then the memory requirements are in unison, since even with the dynamic allocation there still has to be room for those allocations. So the so the total memory requirement is similar to the statically allocated.

Estimations of a few worst-case total memory requirements are seen in Table 2. For 32 users, 6 HARQ processes each and maximum 28800 bit TB size would be  $32*6*28800b = 691.2$  kilobytes. This amount would be needed only in theoretical situations and the amount of simultaneous users could easily be dropped to 24-20 without affecting the actual performance of the HARQ.

Table 2. HARQ memory requirements

Simultaneous users	32	24	16	8
HARQ processes per user	6	6	6	6
Transport Block size in bits	28800	28800	28800	28800
Required memory in kilobytes	675	506.25	337.5	168.8

With a maximum of 24 users supported by HARQ the HSDPA implementation could still support 32 simultaneous users because, in practice, everyone will not require transmission bandwidth simultaneously. Another factor is that Transport Blocks are buffered into HARQ only when they are scheduled for transmission. Only a limited amount of users can be served per TTI and then there is the fact that typically TBs have to be buffered only for a few TTIs before discarding. Therefore the HARQ with support for 24 or even 20 HARQ entities is approximated to be enough.

#### ***2.4.6. Adaptive modulation and coding***

The idea of Adaptive Modulation and Coding (AMC) is to change the coding and modulation rates with the changing channel quality. AMC increases the capacity to adapt to the prevailing channel conditions. AMC, also known as Link Adaptation (LA), is responsible for selecting the correct Transport Format and Resource Combination (TFRC) for the data in current channel conditions. It also has to take

into account the UE capability in demodulation and the available code and power resources for HSDPA in the cell. [6]

A link adaptation scheme makes possible to the user to benefit from more advanced receiver structures with increased data rates. Traditionally CDMA systems uses performance gains of a more advanced receiver to increase capacity in terms of increased users and doesn't benefit the user with increased QoS e.g. in terms of increased data rate. [7]

AMC makes its channel estimations based on the feedback from the UE. Either the associated DCH channel closed loop Transmit Power Control (TPC) commands sent by the UE in the uplink DPCCH, or from the Channel Quality Indicator (CQI) messages received via HS-DPCCH from the UE can be used for the channel quality estimation. An algorithm produces an optimal TFRC estimation per TTI for each UE that has data in the scheduling queue. TFRC report sent to the UE consists of modulation type information, number of multicodes (1 to 15 HS-PDSCHs) and the transport block size. [6]

Adaptive modulation and coding offers several challenges, must not be overlooked. AMC is very sensitive to any measurement errors and delays. To produce the TFRC the algorithm has to be aware of the channel quality and errors in this estimation will result in the selection of wrong modulation and coding methods. Delays in the channel measurement reports also reduce the quality of the estimate due to the constant variations of the mobile channel. HARQ reduces the AMC sensitivity to errors and traffic fluctuations and makes possible the implementation of AMC with a less number of modulation and coding schemes. [12]

#### ***2.4.7. Architecture aspects***

The greatest architectural change that the HSDPA introduces to the Rel. 99 is the new MAC-hs entity on the Node B. The reason for this is that the previously described features of HSDPA rely heavily on the rapid adaptation to the changing channel conditions. To be able to adapt quickly the decisions have to be made with minimal delay and therefore the functionality should be placed as close as possible to the air interface, in Node B. Lots of functionality like ciphering and in-order delivery of data still remain on the RNC and the expanded capabilities of Node B should be seen as an extension or complement from the RNC point of view and not as a replacement of the RNC. [7]

#### ***2.4.8. Performance***

The exact data rates for HSDPA transmission depend on so many variables between the transmitter and receiver that only a small example is given here. An example of possible data rates and their relations is given in Figure 18, which displays data rates achieved with different modulation options, code channel amounts and code word strengths.

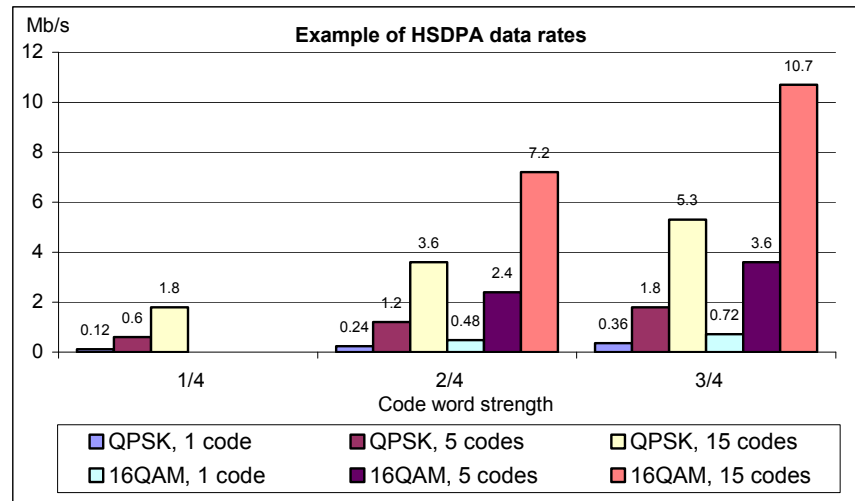


Figure 18. Examples of HSDPA data rates with QPSK and 16QAM modulations.

From the figure one can see that 16QAM modulation provides superior performance over QPSK, even with a more redundant coding and with less or an equal amount of code channels. QPSK modulation can support a maximum of 5.3Mb/s and 7.2Mb/s when 16QAM with  $\frac{3}{4}$  coding and 15 code channels can support over 10Mb/s. [13]

#### 2.4.9. Requirements for memory management

HSDPA requires significant amounts of processing power and memory space. HSDPA is a difficult thing for the memory management because of the high amounts of memory traffic required and because it's traffic is very bursty. The typical applications and fluctuating network conditions create bursts in the traffic, but also the operating idea of radio link and packet scheduler create high instantaneous bursts of data traffic from the base station to the user.

Figure 19 shows the operation concept of HSDPA without physical layer components. In the figure there are four components: RNC, HSDPA-DSP, decoding & channel info and encoding. RNC has it's own flow control and HS-DSCH frame protocol functionalities. MAC-d PDUs are sent via the frame protocol from the RNC flow control to the HSDPA flow control residing in the Node B when the HSDPA-DSP requests. The HSDPA-DSP has to handle the HS-DSCH Frame Protocol or have a serving component that extracts the MAC-d PDUs for it.

Flow control has to buffer these PDUs for each user in their corresponding priority queues. In each queue there has to be data for at least 10 TTI, depending on the round trip time on the DSCH FP signal path. Previously calculated memory amounts show that for 16 users with 6 priority queues there should be 3375kB of memory. The capacity allocation method should be carefully considered with costs versus benefits since a complex method is going to take a high toll of the DSP clock cycles.

The Packet Scheduler selects the UE to serve based on the selected scheduling algorithm. Whatever the algorithm, it has to be fast and simple to be able to select the scheduled user in the 2ms time. PS then informs the selected UE via the HS-SCCH

channel, carrying all the transmission parameters, that there is an incoming HS-DSCH transmission. It also assembles the Transport Block from the MAC-d PDUs buffered in the Flow Control. TB is then sent for HS-DSCH encoding. The Packet Scheduler is one of the key components ensuring the effective operation of HSDPA base station.

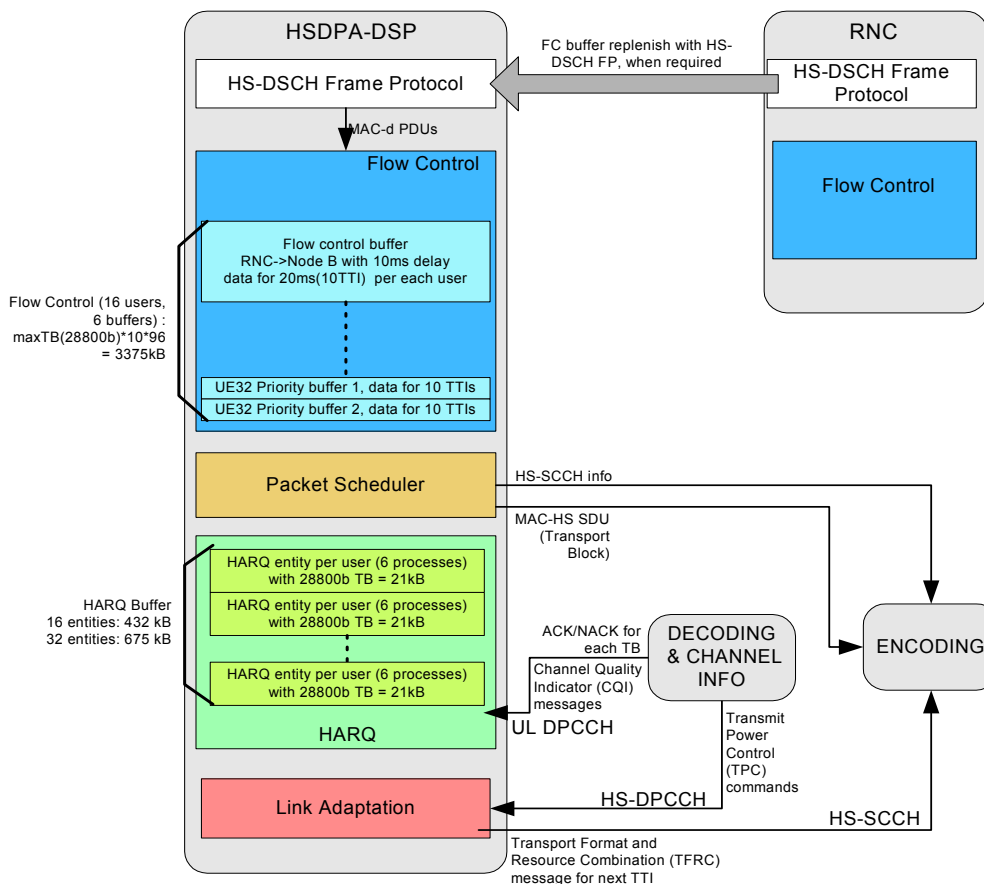


Figure 19. HSDPA Operation concept.

PS assembles the TB for encoding and transmission and also buffers the TB with a HARQ process, which creates a new entity for the task. Each HARQ process buffer one TB into the memory assigned to it. The memory required by HARQ depends on the number of simultaneously served HSDPA users. A reasonable estimation for the memory amount assigned to HARQ would be around 337 kilobytes. In theory this will provide support only for 16 users, but in practice it will most likely be enough for at least 20 users, depending on the capabilities of the memory management. Sizes of Transport Blocks that are buffered with HARQ are seen on the Appendix 1. From the listing in the appendix it can be seen that the variations in the TB sizes are large. HARQ also monitors the ACK/NACK and CQI messages from UL DPCCCH channel.

HSDPA-DSP has also a Link Adaptation component to monitor incoming TPC commands from HS-DPCCH and to select the correct modulation and encoding parameters for the next TTI. These parameters are transmitted to the UE via HS-SCCH in the TFRC signal.

The Link Adaptation and Packet Scheduler together consume only a minor portion of memory and mostly they use the stack. HARQ and Flow Control, on the other hand, are both feature very heavy and frequent memory activities. So the estimated total memory requirement for HSDPA-DSP would be 3807kB RAM and about few hundred kilobytes for program code. Approximately four megabytes of memory would be enough.

### **3. BASE BAND DIGITAL SIGNAL PROCESSING ENVIRONMENT**

This chapter introduces the base band digital signal processing environment. The chapter begins by presenting the typical Digital Signal Processor (DSP) environment structure and then moves on to give an understanding of the hardware. Finally there is a section for comparing the performance of different processor architectures. It is important to illustrate the challenges presented by a typical DSP architecture to memory management. The knowledge of the features of the executing hardware is important for understanding the results of the memory management experiments done in chapter 5.

#### **3.1. Introduction to DSP processors**

Digital signal processors can be divided into two principal groups: programmable DSP and Application Specific Integrated Circuits (ASIC). Programmable DSPs are just what their name implies: digital signal processors that can be programmed and their signal processing functions are implemented with software. ASICs are integrated circuits designed for a specific application and DSP functions are implemented with hardware since ASICs cannot be reprogrammed. This thesis does not deal with DSP ASIC and therefore any references to DSP should be understood as programmable DSP.

DSPs are specialized embedded processors that target low cost, low power consumption, small code size and real-time response. These same specifications are also targeted by microcontrollers (MCU). MCUs and DSPs have the same goals, but different architectures and they are specialized in different types of operations. The key architecture features of DSPs include multiple multipliers and arithmetic logic units, address generation units, multiple memory accesses per clock cycle, optimized instruction set and usually no hardware memory management units for cache nor main memory. DSPs favor tight computationally demanding loops, highly predictable data stream processing and accessing data in an orderly manner. MCUs are at their best when dealing with random events and frequent conditional branches. [14]

A typical DSP is targeted to exploit over 90% of its gross computing power. If some application requires less, then the clock rate of the processor can be decreased. This results in power savings, which are so important in any portable device. They can usually perform multiple multiply-accumulation (MAC) operations and complete several memory accesses in one instruction cycle. They are specifically designed for computing intensive applications where traditional general-purpose processors are too slow and custom ASIC is too expensive or otherwise unfeasible. DSP processors are used in a variety of applications ranging from toys and consumer electronics to military radars and scientific sensors. In general they are best suited for products supporting several standards or handling multiple functions and especially if upgradeable solution is required. [15] [14]

A DSP is almost never used alone. An example of an typical environment for a DSP is illustrated in Figure 20 where a DSP is combined with an application specific

integrated circuit, external memory, programmable gate array and a microcontroller to perform the control functions.

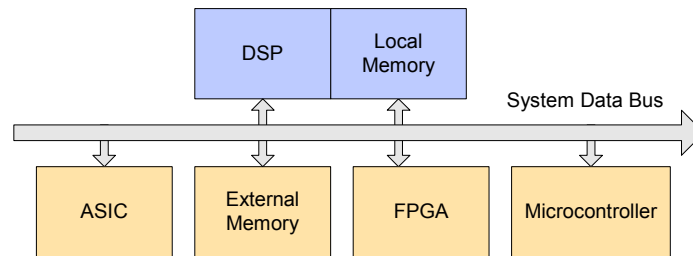


Figure 20. Typical DSP Environment.

Usually these components are interconnected with a system data bus capable of high-speed data transfer and control operations. Typically in this kind of DSP environment the DSP itself is a tool for the controlling microcontroller to perform intense computing operations that it itself is incapable of completing in the required time.

### 3.2. WCDMA Node B

The Node B is a network element in the mobile cellular network that handles the radio communication with the UE. A more detailed description of the network components and Node B is given in chapter 2. The principal structure of a Node B is presented in Figure 21. When the antennas receive the signal it is passed to the AD conversion via duplex filters and low noise filters. After analog-to-digital conversion the signal is passed to L1 signal processing and channel decoding to the base band unit. In the case of HSDPA transmission the base band unit also handles the MAC-hs protocol, otherwise the MAC protocols would not come into the picture before RNC. After base band unit the data is passed to the RNC via ATM connection. In the transmission process the same path is traveled, but in the opposite direction.

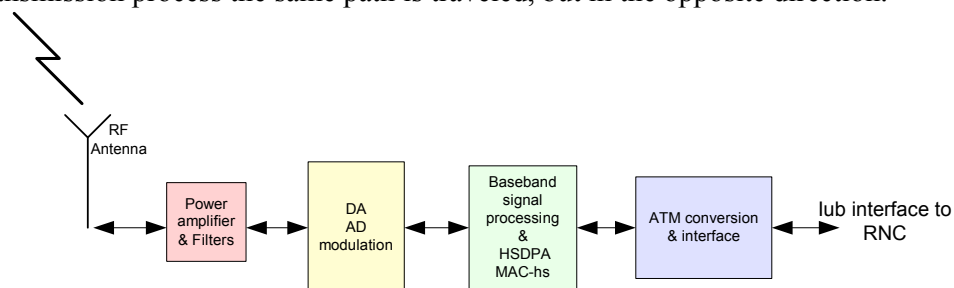


Figure 21 General block diagram of WCDMA Node B architecture.

A Base band unit is used for Layer 1 and Layer 2 signal processing in uplink and downlink directions in the Node B. The base band unit operates with Texas Instruments C6000 –series DSPs with up to 4MB external memory. In addition to DSPs the unit also has a microcontroller unit (MCU) and some custom ASICs. The general block diagram of the base band unit is displayed in Figure 22.

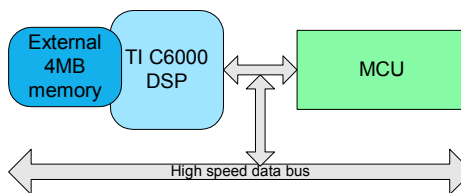


Figure 22. Block diagram of HSDPA DSP environment.

HSDPA is implemented in this kind of environment seen in Figure 22. Both the L1 and L2 signal processing are done with the same kind of DSPs. MCU provides controlling functions and some extra processing capability.

### 3.3. Processor performance comparison

This section compares three different processors and their features, special attention is given to their branch handling methods. Branches in all their different forms characterize memory management operations and are especially costly for DSP processors. Therefore the abilities of a Texas Instruments TMS320C67x DSP, Motorola G2 PowerPC low-power microprocessor and AMD Athlon processor are compared. Comparison is done to illustrate the architectural differences in the different processor types and to emphasize the primitivism of a DSP. However, the primitivism can also be seen as an optimized design, which depends on the viewpoint. TI's C67x represents a typical DSP with none, or almost none, branch handling features. Motorola G2 represents a high performance microprocessor from the embedded market with some branch prediction features that allow it to execute branches in certain situations with zero latency. AMD Athlon represents a typical modern general-purpose microprocessor found in desktop PCs.

#### 3.3.1. Texas Instruments TMS320C67x

Texas Instruments TMS320C67x –series core has a pipelined Very Long Instruction Word (VLIW) architecture. The pipeline is seen in Figure 23 and it is divided into three stages: fetch, decode and execute. The fetch has four phases, decode uses two and execute has ten. Every instruction goes through all phases of the fetch and decode stages. The amount of execute phases varies by instruction. Generally all integer operations are faster than floating-point operations and multiplication is faster than division.

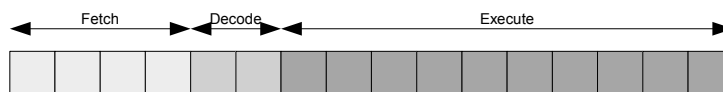


Figure 23. TMS320C67x pipeline.

In the figure one block depicts one processor clock cycle. The pipeline can dispatch eight parallel instructions every cycle. Parallel instructions proceed together through every phase of the pipeline.

During the fetch stage the C67x fetches a packet of the eight next instructions. These instructions are then decoded and passed to the appropriate functional units. It has two multipliers units and six arithmetic-logic units. The C67x also features support for single- and double-precision IEEE floating point operations. It also features conditional execution of all instructions to minimize the effects of costly branches and to increase parallelism. However, conditional instructions are a viable solution only for very small loops and if-statements that the compiler can effectively unroll and schedule for execution. A branch takes one execution phase and has five delay slots. [16]

### ***3.3.2. Motorola G2 PowerPC 603e microprocessor***

The Motorola G2 CPU is a derivative of a MPC603e PowerPC microprocessor design. The G2 is a 32bit low-power Reduced Instruction Set Computing (RISC) microprocessor with a superscalar design that can issue three parallel instructions per clock cycle. The processor has five execution units: an integer unit (ALU), a IEEE compliant floating-point unit, a branch prediction unit, load/store unit, system register unit and a completion unit, which makes it possible out of program order execution and predicted instructions. The five execution units make possible for five instructions to be executing simultaneously, but only three instructions can be issued or completed per cycle. The floating point and integer units are pipelined. There is also a 16kB cache for instructions and data. The processor supports virtual memory addressing with its memory management units.

The G2 Branch Prediction Unit (BPU) is an effective way to minimize the costs of branches. When a branch instruction is fetched, the BPU tries to pull the instruction out of the instruction stream and resolve it. The following instructions are then moved on to fill the gap left by the branch instruction. If the branch is not taken then the execution of instruction flow continues and the branch has been processed with a zero cycle delay. If the branch is taken and the branch target changes the instruction flow, the BPU requests new instructions from the instruction cache. If the cache request is a hit, then the instructions are readily available with only one cycle delay, which is usually not enough to create stalls in the execution units. But if the cache request is a miss then there is a six-cycle delay with the instruction fetching. This whole technique is called branch folding. BPU has also a feature for minimizing the costs of branches: static branch prediction.

If there is a data dependency between the branch and some previous instruction still in execution, and the branch cannot be resolved, then the branch is predicted with a static branch prediction. Normally the execution and instruction fetching would have to be stopped until the result of the instruction on which the branch depends, is computed. In static branch prediction the compiler tries to predict the result of this kind of branch and codes this information to the program. BPU then looks for this info, predicts the result of the branch and orders instructions to be fetched and executed along that direction. If the prediction was correct, then the program flow continues uninterrupted and the branch has been resolved with zero delay. If the prediction is incorrect the wrong instructions are flushed from the execution units and new instructions are loaded. The amount of instructions that can be executed after branch prediction is limited by the fact that these predicted

instructions are not completed and allowed to update memory and register contents, until the result of the branch is resolved. [17]

### ***3.3.3. AMD Athlon***

The Athlon belongs to the AMD K86 microprocessor family that is binary software compatible with the Intel x86 architecture. The x86 architecture is known as Complex Instruction Set Computer (CISC). The Athlon uses an architecture that breaks up the variable length complex instructions to fixed size simple operations, which can be executed, out-of-order when their operands are ready. The Athlon is a 32-bit is a superscalar processor that can fetch, decode and issue up to three parallel x86 instructions per cycle. It also features dynamic scheduling and speculative execution of instructions. It has two instruction schedulers, one for the integer unit and one for the floating-point unit. These schedulers can issue up to nine simultaneous operations. The Athlon has three integer ALUs and three floating-point units with support for 3Dnow! and MMX instruction extensions. Integer units and floating point units have their own pipelines. The integer pipeline execution stage has five phases and floating-point execution has nine phases.

The Athlon has Level 1 2-way associative instruction and data caches of 64kB sizes and two level Translation Look-aside Buffers for paged virtual memory support. Level 2 cache size is typically 512kB, but there is support for up to 8 megabytes. Older models have a 2-way associative L2 cache, but newer the models have a 16-way associative cache.

For branch prediction the Athlon uses a combination of a branch target address buffer (BTB), a global history bimodal counter (GHBC) table and a return address stack (RAS) hardware to predict and accelerate the execution of branches. The BTB is a 2048 entry table where addresses of each branch are cached with their target addresses. Bimodal branch prediction tries to find the typical behavior of a branch and records the previous branch results. It makes the prediction based on the direction the branch went on the last few execution times. RAS improves unconditional branch prediction by keeping a stack record of the function call return addresses. The penalty cost of mispredicted braches is 10 cycles minimum. Correctly predicted branches have only a 1 cycle delay. [18]

### ***3.3.4. Summary***

All three processors presented here are just examples of their kind and are designed for different kind of applications. It is important to understand the drawbacks of a DSP. Table 3 presents a comparative summary of the processors. By comparing the cache and branch minimizing hardware features of a DSP and the two other processors the general primitiveness of a DSP can be seen. A DSP has many more functional units and can issue over two times more instructions per cycle. This shows the idea of maximizing the execution throughput and optimizing for tight compiler predictable loops with parallel multiply-accumulation operations. Also the lack of a MMU for paged virtual memory in DSP should be noted. From the memory management viewpoint the interesting thing is the lack of branch minimizing hardware features. Only conditional instructions are supported, which will lead to a

high toll in every branching situation. This emphasizes the importance of optimized code and the need of finding a light memory management method. Otherwise the execution of the memory manager will take an unnecessary high percentage of processor cycles while resolving branches in allocation requests.

When comparing the computation operations things start to look better for the DSP. It can theoretically compute 1800 million integer operations per second and 600 million FP ops. The G2 can execute 300M integer ops and 300M FP ops. The Athlon can compute 4200M integer ops or 4200M floating ops. These numbers are just theoretical figures, which are never reached in real life situations.

Table 3. Processor comparison

	<b>TMS320C67x</b>	<b>G2 PowerPC</b>	<b>Athlon</b>
Functional units	2 FP units, 6 ALUs	1 FP unit, 1 ALU	3 FP unit, 3 ALUs
MMU	-	Yes	Yes
Clock frequency	Max. 300 MHz	Max. 300 MHz	Max. 1400 MHz
Cache	-	16kB data, 16kB instruction, 4-way set-associative,	L1: 2-way assoc. 64kB data, 64kB instruction L2: 512kB – 8MB 2-way or 16-way assoc.
Instructions per cycle	8	3	3
Floating point latency	DP mult. 10 cycles, SP mult. 4 cycles	DP mult. 4 cycles, SP mult. 3 cycles	DP mult. .6 cycles, SP mult. 6 cycles,
Branch minimizing HW-features	Conditional instructions	BPU for branch folding and static branch prediction	2048-Branch Target address buffer, Global History Bimodal Counter, Return Address Stack and speculative execution
Branch penalty	5 cycles	Correct prediction 0, Incorrect prediction 6	Correct prediction 1, Incorrect prediction 10+
Branch prediction accuracy	0%	65%	>90%

Effectiveness of static branch prediction hardware is hard to tell without experiments. By comparing the processor branch prediction comparison table found in [19], the efficiency of the branch prediction of the AMD Athlon processor is estimated to be around 90% or a little over. Static branch prediction mechanisms like those found in the G2 processor have usually 60-70% efficiency in correctly

predicting branches. It can safely be estimated that the G2 is at least 100% faster at executing heavily branching code than the C67x. [19]

Table 4 shows the approximated amount of used cycles for each processor for a certain number of branches. The number of branches are presented in the columns on the right side of table: 15, 50 and 100, respectively. For 15 branches the C67x uses 75 cycles, when the G2 uses 31.5 and the Athlon 25.8 cycles. This very synthetic example is based on estimation of the prediction accuracies and doesn't take clock rate into account.

Table 4. Branch execution times in clock cycles

Processor	Branch prediction accuracy	Penalty of a predicted branch	Penalty of a mispredicted branch	15	50	100	Speed advantage to C67x	Clock rate normalized advantage to C67x
C67x	0	5	5	75	250	500	0%	0
G2	0.6	0	6	36	120	240	108%	2.45
Athlon	0.95	1	10	21.8	72.5	145	245%	16.05

The higher the clock rate, the higher are the costs of a misprediction. This is seen as the rather low advantage of the Athlon in terms of cycles, which has much higher clock rate than the two other processors. If clock rates are taken into account then the difference of Athlon and C67x increases even more. Table 4 shows that with normalized clock rate the Athlon can execute branches 16 times faster than the C67x. When comparing only used clock cycles the Athlon is 245% faster than C67x and about 66% faster than the G2. When the G2 and C67x are compared it can be seen that the G2 is 108% faster.

Based on these facts the G2 and Athlon will greatly outperform the C67x DSPs when executing program code containing a high frequency of branches. Program code is like typical control code or memory management code, where comparisons are frequent. The C67x will have to delay its execution on every encountered branch for five cycles. The G2 and Athlon can usually predict the branch and continue the execution with minimal or no delays. A conservative estimate is that memory management code executed with Athlon consumes only 10% of the time it takes on the C67x

No matter how unoptimized the DSP is for memory management tasks, it still has to perform them when operating in a HSDPA base station. The task is to find a method that gives good results and is reasonably fast to execute. In this task both the costs and benefits of each method have to be carefully weighted. The next chapter presents the methods, concepts and the problematic of memory management and its features. It also evaluates the costs and benefits of each feature.

## 4. MEMORY MANAGEMENT

This chapter presents the concepts, problematics, analyzing methods and background for memory management. The chapter begins by introducing the concept of dynamic memory management, followed by some of the common problems and introduction of different allocation policies in memory management. The chapter is concluded with a summary of the requirements for DSP by memory management.

### 4.1. Dynamic memory management

Memory management can be divided into three parts: memory management hardware, operating system memory management and application memory management. Memory management hardware means all the storage systems, circuitry and electronic devices used by the computer to store data. Virtual memory, paging and segmentation are commonly known methods that require memory management hardware support. Operating system memory management operates the memory management hardware and provides areas of memory to the applications. Application memory management then manages the memory provided by the operating system for dynamically changing storage requirements.

A memory allocator program handles the application memory allocation and a recycler takes care of the freed blocks. The task of an allocator is to keep track of the parts of the memory that are in use and of those that are free. And by using this knowledge, coupled with its placement strategy, to respond to any requests from programs to store program objects in the memory. The task of the recycler is to make the freed memory block again available to the allocator. [20]

Dynamic memory management is the management of memory with methods that allow reusing the memory after it's released from a reserved allocation. The motivation for dynamic memory management and allocator study is that poor memory management leads to a loss of memory and processing cycles in computers. Memory that is unusable is just wasted resources that have been acquired with a price. These are hidden costs that are hard to account directly and can lead to great losses in productivity. Or in the worst case the unusable memory and wasted processing cycles may lead to a system malfunction. This chapter focuses mainly on the allocators. [22]

The goal of an allocator is to minimize the lost space without affecting the processing speed and vice versa. An ideal allocator is one that spends an insignificant amount of time managing the memory and wastes an irrelevant amount of space. In DSP memory management it is especially important that the allocator used is as close as possible to this ideal allocator. The reason for this is that the execution time used for memory management is away from the actual signal processing tasks and compensating for any wasted memory shows as higher hardware costs. Another important feature of an allocator used for DSP is that the execution time is foreseeable and preferably doesn't have large variations. The knowledge of the execution time is important for real-time systems, because they must guarantee some worst-case execution time. This time must fit into the real-time constraints for the system to be able to operate in real-time.

An allocator cannot affect the number or size of requested blocks, since these are completely up to the requesting program. It also cannot move the reserved blocks around in the memory to gain contiguous and free memory, unless compaction is used.

In compaction all live objects are moved to the one end of the memory to gain one large contiguous free area of memory. Compaction is a difficult process since all internal addresses of processes have to be updated and the allocator must be able to determine the liveness of an object. [21]

An allocator has to respond immediately to any requests and it cannot change the placement decision once it has been made. The memory block where the object is placed is called a reserved block. A reserved block has to remain untouched until the requesting program decides to free it. The allocator can only deal with free memory and use only the free memory area where to allocate a requested block.

Dynamic memory allocation is typically used with large complex systems, which would be unfeasible with static allocation. When all memory allocations are done with requests to the allocator program, the task of the programmer becomes much simpler. Without dynamic memory management the programmer would have to keep track of all allocations and this easily leads to memory leaks and errors.

Different allocators are designed for different purposes. They are also optimized for different properties like execution speed, internal and/or external fragmentation, handling large memory spaces, wide variety of allocation sizes and fine granularity in allocation sizes etc. The selection of allocator depends on the desired features and in this process the costs versus benefits have to be considered for each property. For base band DSP the desired features are known execution speed and minimum fragmentation. Execution speed means that any exhaustive searches for best possible choices and any complex methods are out of the question.

## **4.2. Memory management problems**

There are several important problems with memory management. The most important of them is the fragmentation, which is discussed in detail in Section 4.4. In brief it can be described as the memory that is free and unused, but the allocator cannot use it, because it is in such small chunks.

Another problem is called poor locality of reference. This means that modern operating systems and hardware can make very fast successive memory accesses if the accessed blocks are in nearby memory locations. If the memory manager scatters the blocks belonging to the same program all around the memory space, far away from each other, it can lead to performance degradation.

A memory manager can be optimized for a certain kind of application, take advantage of the knowledge of the memory access patterns, block sizes and lifetimes of that application. If it is then used in a different environment with different kinds of applications it is typical that the memory management is not as effective and the allocator suffers a serious performance loss. Of course this problem can be avoided by not using specifically tailored managers with a wide variety of programs all with different kind of behaviors. This is known as inflexible design of the memory manager, or it can also be said to be highly optimized design, depending on the situation and viewpoint.

A memory leak happens when allocated memory is not freed after its intended use and this results in that it can be never used again. In manual memory management, this usually occurs because objects become unreachable without being freed. Repeated memory leaks cause memory usage of a process or program to grow and grow without any constraints. This usually leads to a program crash or at worse crashing the whole computer when it runs out of memory. The reason for a memory leak is usually a programmer's error when he forgets to free the memory he has requested. The allocator cannot help in memory leak situation because it does not know whether the allocated blocks are still needed or not. There are methods to determine if the block is still needed or not though, which is known as the liveness of the block. These methods are used in advanced garbage collectors. Garbage collectors automatically recycle the dynamically allocated memory. Even the methods used by the garbage collectors can only give approximations of the liveness status and cannot know exactly which blocks are still live. Advanced garbage collection methods are estimated to be too complex for DSP and in typical applications they are generally unnecessary.[20]

### 4.3. Memory fragmentation

Memory fragmentation is the most important problem with dynamic memory management, which the allocator can effectively combat. There are two kinds of fragmentation: internal and external. Internal fragmentation is the lost memory, which occurs when allocating some size of memory block and the actual required amount of memory for the data is a little bit smaller. An example of this is shown in Figure 24. where a allocation request is for 63 bytes and the allocator can only allocate blocks with size of 128, 64 or 32 bytes. In this case the 63 byte request would be served with a 64 byte block, which would waste 1 byte of memory.

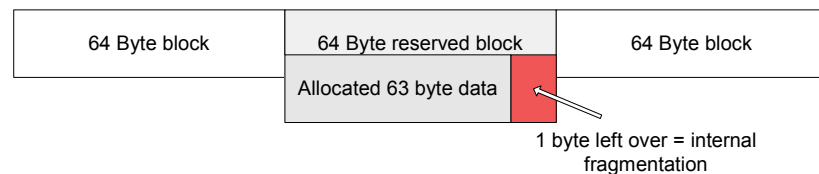


Figure 24. Internal fragmentation.

In this case the whole 64-byte block is marked as reserved and the request is served, but the one extra byte is lost until the block is freed again. This one lost byte would then be accounted as internal fragmentation. [24]

Internal fragmentation can be a very serious problem if the allocator can allocate only a few different block sizes or if the different block sizes have large spacing between them. In this situation the allocator has to satisfy most of the memory requests with too large block sizes. Internal fragmentation will be measured later on in this thesis.

External fragmentation is the type of fragmentation, which is commonly known just as fragmentation. When the memory allocator is unable to use the available free memory because it is divided into too many small blocks, the unusable space is then accounted as external fragmentation. External fragmentation is created when several blocks of memory are allocated simultaneously, but are not freed together and

thereby create small gaps of free memory in the middle of reserved memory. This is illustrated in Figure 25 where there are four successive free blocks and three blocks are then allocated at those positions. Then block 2 is freed and this leaves a gap to the memory and a new allocation for a larger block 4 is impossible. If block 3 had been freed instead of block 2, then the two free blocks could have coalesced to satisfy the request for a larger block 4.

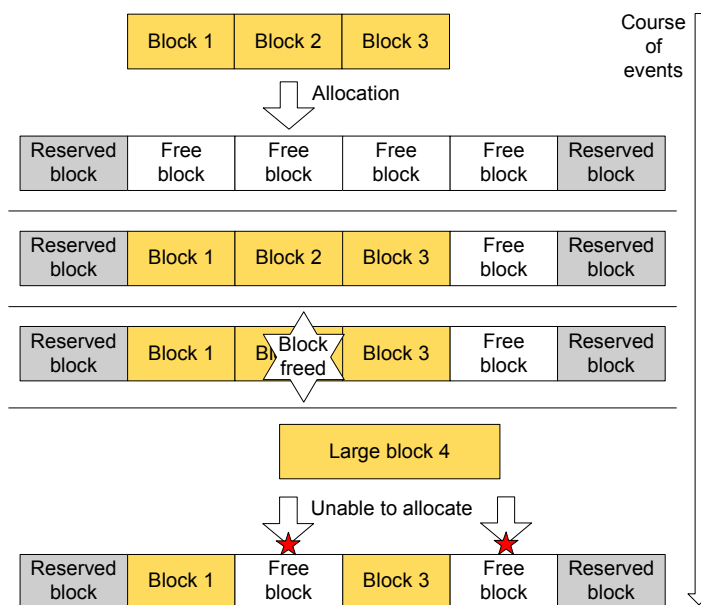


Figure 25. External fragmentation

This kind of situation could occur if three successive blocks are allocated and then the one in the middle is freed. It is possible to reuse the gap, but it is not anymore possible to allocate a block as large as all the free memory. [24]

External fragmentation keeps a system suffering from it from using its full memory space efficiently. If the amount of fragmentation is not reduced at any time it might result in the total crash of the system. Some allocators produce more external fragmentation than others. In chapter 5 a few allocators and their tendency to produce external fragmentation are measured and studied.

External fragmentation can be reduced by using compaction, which was briefly explained in the Section 4.1. On a base band DSP compaction could be run intermittently, for example during times when there is little or no user activity. A difficult part of compaction is to update all the pointers to the changed memory allocations. A solution for this is to use handles. Handles are objects, which store addresses to allocated memory blocks. Memory could be accessed through these handles and the compaction procedure would only have to update the memory addresses in the handles. From the programmers view the handles can be understood as a pointer to a pointer. Another option is to make all memory blocks the same size. This way any fragments, holes of free memory, would always be usable. The downside of this is that if the allocated data is not of the same size there will be much internal fragmentation. However, this idea of using fixed size blocks to minimize fragmentation is usable with HSDPA Flow Control component that allocates only blocks of the same size.

#### 4.4. Memory allocation policies

When designing a memory allocator it is very useful to take advantage of regularities in the application program behavior to prevent fragmentation and make the allocator work as well as possible. As the allocator has many possible locations in the memory where to place the requested block, there has to be some general rule, or strategy, in this placement policy. The allocation strategy attempts to take advantage of the known regularities in the memory requests. A policy is the implemented decision procedure for placing blocks in the memory, and a mechanism is a set of algorithms and data structures that implement the policy, often over-simply called “an algorithm” [22].

All allocators have their own kind of data structures used to keep track of the reserved and un-reserved memory blocks. Each of these data structures requires some amount of memory by themselves. If the allocator stores this data into the memory so that it takes away space from the actual data, it is called overhead. Overhead can be understood as the memory costs of book-keeping required to operate the allocator. Another kind of overhead is the number of clock cycles used to run the allocator.

##### 4.4.1. Common functionalities in all policies

Almost all allocators are implemented with splitting and coalescing abilities, which are important subsidiary parts of any allocator implementation. Coalescing is portrayed in Figure 26. In the figure two small 4kB blocks are coalesced into one 8kB block. With coalescing the allocator can merge two smaller free blocks of memory into a one larger one. When a block is freed the allocator usually checks if either of the neighboring blocks are free for coalescing. It is recommended to use coalescing if possible, since one large block is typically likely to be more usable than two smaller ones.

Coalescing can also be used in deferred mode where coalescing is done only intermittently. This is because typically the block sizes requested from the allocator are the same as freed before and frequent coalescing would waste precious processor clock cycles unnecessarily. If every possible block is coalesced, a lot of work would be wasted when the block has to be split again into the previous size when serving a request with a same block size as with the previously freed block. Coalescing is an effective method for combating against external fragmentation, which is described in Section 4.3.

Coalescing is quite simple to implement as needs only to compare the status of its two neighbors and update the pointer and status information, but these are not done very effectively on a DSP. Deferred coalescing would be preferred for DSP.

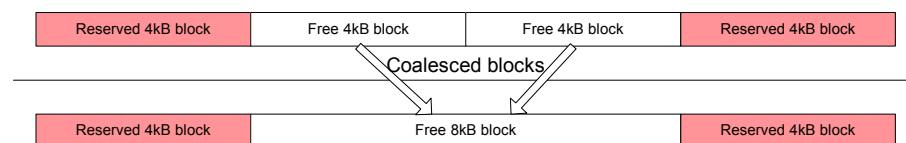


Figure 26. Coalescing blocks of memory.

The principle of splitting is displayed in Figure 27. In this figure the allocator splits one free 8-kilobyte memory block into two 4-kilobyte free blocks. The ratio of the split blocks can be chosen arbitrarily in each case, it doesn't have to be  $\frac{1}{2}$  as in the example in Figure 27. With splitting the allocator can split one free block of memory into two smaller ones. [22]

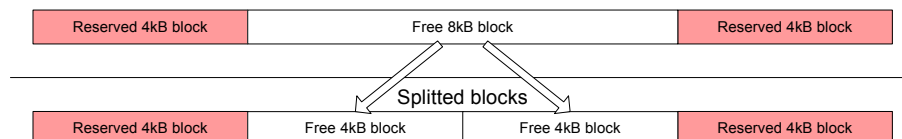


Figure 27. Splitting a block of memory.

Most allocators implant a header field into each block to store information like the size of the block, flags if it is in use or not, neighborhood relations etc. The most common information is the block size. Typically the size of the header is 32 bits or similar. That is enough for holding the size of the block and leaves room for a few flags. Improved versions of the header fields are called boundary tags. They have also a footer field and the end of the block. They both hold record about the block, similar information as described earlier. Boundary tags become useful when the block is freed. It is very easy to check the footer and header fields of the adjacent blocks to see if they are free to be coalesced to form a larger free block. In addition to the header field in each block, some memory space is required usually from the beginning of the whole memory area of the allocator to store the required information for the operation of the allocator. It is a sort of a master header.

These kinds of fields come with a price; the memory used for boundary tags is not available for data. This overhead can be avoided with an optimization that removes the need for footer fields in blocks that are in use. The idea is seen in Figure 28.

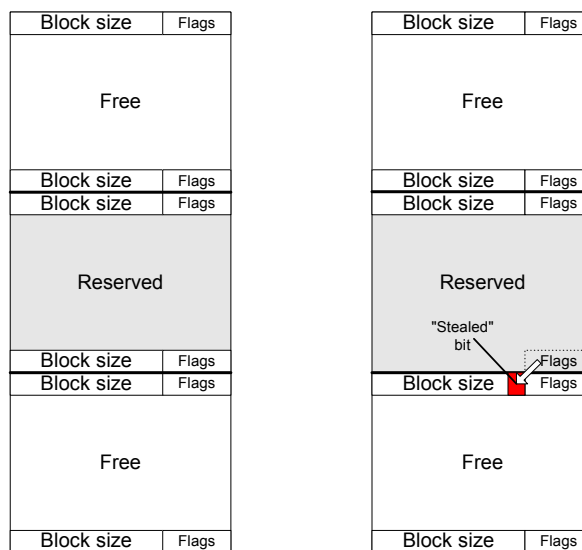


Figure 28. Boundary tags and their optimization.

When the block is in use the only thing needed in the footer is the flag saying that the block is in use and unavailable for coalescing. This one bit flag can be placed into

the header field of the following block without many limitations on the block size ranges. In the figure this is shown as the stolen bit and colored in red. [22]

For those allocators using some kind of tree structure to keep track of the free blocks, a useful method is to store the data structure into the free blocks themselves. This way there isn't any lost space due to the allocators book-keeping purposes. Each free block can contain a pointer or pointers to the next free index or node in the free lists. The drawback in this is that if each free block holds some kind of data structure then there must be some minimum block size. In a case of a smaller requested block the allocator must round up the request to the minimum size.

#### 4.4.2. Sequential fits

The Sequential fit method keeps their free memory blocks in a single linear list that is double-linked. When a memory request is received the list is checked through according to the method in use. Figure 29 illustrates the linked list data structure of a sequential fit method. If suitable memory blocks are found, they are reserved for the requester. When the memory is not needed anymore, the blocks are added back to the list for reuse. [23]

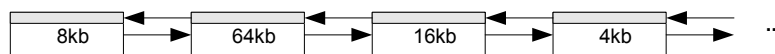


Figure 29. Double-linked list of Sequential Fits.

Best fit allocators search the free list to find the smallest block to fit the requested size, hence the name best fit. The method attempts to minimize the amount of wasted space, but there is the danger that the remainder might be too small for any actual usage. Generally the method is exhaustive, though it stops if a perfect fit is found. This results in the fact that the best fit doesn't handle large heaps with many free blocks well, because it is so slow to search the best fit from a large memory space. The Best fit allocator has a tendency to create many small holes in the memory, external fragmentation. [22]

First fit allocators simply search the list from the beginning until a first block that can accommodate the requested size is found. If the block found happens to be larger than necessary it is split and the remainder is then added back to the free list. It is this splitting behavior that creates a major problem for the first fit method. It has a tendency to split the large blocks at the beginning of the list and after a while the result is a large amount of small blocks at the beginning of the list. If first fit is used without the described splitting behavior, it then has a tendency to create a high amount of internal fragmentation. This is because the allocator chooses the first block large enough to hold the requested amount of data. Fine-tuning the ordering of the list with different methods like address ordering, LIFO and FIFO ordering can produce considerable gains in the list search speeds. [24] [22]

The next fit allocator is a common optimization of the first fit method. It uses a roving pointer for allocation and this pointer records the position where the last search was satisfied. The next search then begins from there. This method attempts to avoid the pitfalls of the first fit and its tendency to gather small blocks at the beginning of the list. [23]

Sequential fits are simple to implement and execute, but in general do not handle large amounts of memory very well. The data structure is typically a double-linked list and when the amount of memory is high, the search procedures can consume much time. Sequential fit allocators are best suited for small memory amounts where the size of the search list is not very big. Depending on the variation, sequential fit allocators typically produce quite high levels of internal and external fragmentation.

#### 4.4.3. Segregated free lists

Free blocks are ordered by their size in an array of free lists where each list holds only free blocks of a single size. Memory requests are served from the appropriate list and when a reserved block is freed the block is inserted in the list of its size. An example of segregated free lists is seen in Figure 30 where there are four free lists holding block sizes 2, 4, 16 and 32 with various amounts of free blocks.

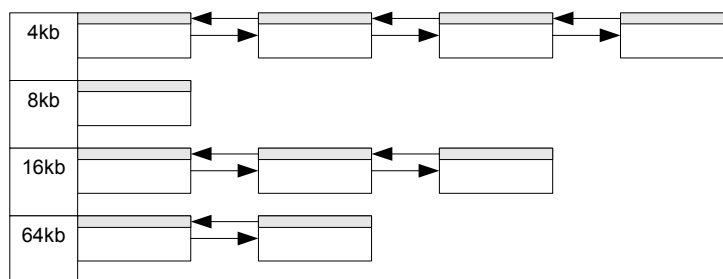


Figure 30. Segregated Free Lists.

Lists are commonly sized to use the power of two in their sizes and round the requested size up so that it matches the closest class, a so-called good fit. This can create internal fragmentation and it is recommended to use a closer list spacing, if possible. The good side of known list spacing is that the maximum amount of the internal fragmentation can be determined beforehand and the list sizes can be optimized to minimize it. The main advantages of segregated free lists are fast memory reserving and freeing because only the right list has to be found. Another key advantage is the already mentioned deterministic internal fragmentation. [23]

Segregated free lists can be effectively optimized for a certain application by studying the sizes that the program typically requests. When these sizes are known, the list sizes can be selected to best accommodate the sizes that the allocator is most likely to encounter. This kind of static optimization is effective if the variety of the sizes is not very large. Another kind of enhancement would be using dynamic list sizes. This way there would be a set of lists without any fixed block size and the block size of the list would be determined when the list is used for its first allocation. If the list becomes empty again it could be resized again if required. This kind of dynamic segregated free lists method would provide the same deterministic internal fragmentation as the original segregated free lists but with much higher adaptivity to different allocation sizes.

#### 4.4.4. Buddy systems

The Buddy systems are a variation of segregated free lists. The whole memory area is divided into two parts and these parts are linked as buddies. Then these two areas are in turn divided into two to smaller areas and linked to form new buddies. Figure 31 shows the general idea of the buddy systems. Each block is split and the resulting smaller blocks are linked to each other, forming levels.

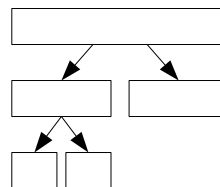


Figure 31. Principle of Buddy systems.

In buddy systems each free block can only be coalesced with its buddy, which is always at the same hierarchical level with it. Each allowable block size is registered in it's own free list, so buddy systems are actually a special case of segregated free lists. If the free list of a desired block size is empty, then the search goes to a higher level where a larger block is split into new buddies with the desired block size.

The idea behind buddy systems is that the buddy for a block of memory is very easy to find with a simple address computation. Another reason is that buddy allocators usually keep track of the blocks in a more advanced data structures than linked lists. It is the known properties of these data structures that make the usage of buddy systems beneficial. Some examples of the buddy systems that use advanced data structures are Binary buddies, where sizes are power of 2, and Fibonacci buddies, where the Fibonacci series is used for splitting. [24] [22]

### 4.5. Summary of memory management for DSP

Memory management for DSP is tricky, because the operations made by memory management are not performed well on a DSP. A memory manager used with a DSP should be:

1. Fast to execute and thus it must use a simple method.
2. Deterministic so that the execution times and especially the worst-case execution times of the allocator can be known beforehand.
3. Minimize fragmentation, both internal and external, to avoid wasting precious system resources. Fragmentation minimization usually requires more advanced allocators, which are more complex and use more clock cycles.

The reason for the first requirement is that basically the memory management is a maintenance and support operation that must not consume processing power from the actual application. The reason for the second requirement is that typically the DSP software is tightly scheduled for maximum throughput. For this the execution times

of different tasks must be known. The reason for the third requirement is that the DSPs have limited memory features, including addressing spaces, and fragmentation increases the actual used memory space. Increasing the amount of memory must compensate increased usage of memory space and this isn't always possible with a DSP. Not to mention that it costs money.

Memory management task for DSP has one particular advantage. The allocation patterns and block sizes are usually well known, because the DSP typically executes the same program at the whole time. This knowledge can be used for optimizing the allocator to perform exceptionally good with the specific task. In a general use such an allocator would be much less effective, but that doesn't matter because it will only be used in the environment and DSP that it is optimized for.

Sequential fit methods are simple, but do not perform well when theoretical worst-case allocation times are considered. Actually observed allocation times are quite much different, which Puaut shows in her study of Real-Time Performance of Dynamic Memory Allocation Algorithms [25]. Buddy systems do better with the worst-case allocation times, but their average allocation time is quite bad.

Segregated free lists are also a quite simple method since only the right list has to be found. This reduces the amount of required comparison operations. If list sizes are too far away from each other it creates extensive fragmentation. This can be avoided by using close list spacing, optimizing the list sizes to the most probable allocation sizes and having a large amount of lists. As a downside a large amount of list sizes results in longer search time for the correct list with the typically used good-fit method. Segregated free lists could be optimized so that the list sizes would be decided dynamically on the run time based on the requested block sizes. This method would reduce internal fragmentation, but would bring a lot more complexity to the implementation. From the DSP viewpoint the costs for this method could be higher than the benefits.

Puaut has measured the performance of some of the same allocator policies as presented here. In Table 5 are listed the results of her study. The numbers in the table are processor cycles. In the experiment she used a Pentium II computer and a 4-megabyte heap for the memory. The processor architecture is wholly different for a typical DSP so the results are indicative at best. The worst-case and mean results are from an allocation trace gained from mpg123 program. The Analytical result is theory-based and calculated by Puaut. [25]

Table 5. Puaut's table of allocation performance

	worst-case	mean	analytical worst
first-fit	6286	6239	115014229
best-fit	7012	6954	117573562
quick-fit	67081	10805	481267
buddy-binary	46167	7299	57827
buddy-fibonacci	39740	10624	47320

Puaut's research shows that sequential fit methods are faster than the segregated free lists (quick-fit in the table) or buddy methods. A very great difference is seen between the actual empirical results and analytical results of sequential fit methods.

The differences in worst-case performance should also be noted since that is an important factor in a real-time system.

In [23] Johnstone and Wilson have measured the fragmentation of several allocation policies that are here presented in Section 4.4 and subsections. They have used actual traces from programs like Espresso, GCC, Perl and Ghost. Table 6 shows the results of their measurements. Their measurement method provides results for fragmentation that is counted in this thesis as internal fragmentation. From the table it can be seen that Address Ordered (AO) sequential fits resulted the smallest fragmentation results with Lea's allocator.

Table 6. Fragmentation results

Allocator	Average
best-fit AO	2.28%
best-fit FIFO	2.30%
best-fit LIFO	2.23%
Fist fit AO	2.30%
first fit FIFO	3.14%
first fit LIFO	36.60%
next fit AO	8.04%
next fit LIFO	38.40%
next fit FIFO	18.40%
binary buddy	53.40%
double buddy	34.30%
Lea 2.6.1	2.28%
simp. Seg $2^n$	73.60%
simp seg $3 \cdot 2^n$	61.50%

In the table are seen results from sequential fits, buddy systems and segregated free lists. Segregated free lists are measured with list sizes of power of two and three times it. Lea is also a segregated free lists allocator with 128 size classes. Lea's allocator does much better in the experiments than the other simpler segregated free list allocators. It yields almost the lowest average fragmentation results.

Both buddy systems resulted in quite high fragmentation results. They are placed between the simple segregated free lists and sequential fits in terms of internal fragmentation.

## 5. MEMORY MANAGEMENT EXPERIMENTS FOR BASE BAND DSP

There are several possible approaches to evaluate the performance of memory management. These include analytic modeling, simulation and prototype implementation. Simulation is a popular approach if the evaluation target is not available, since it provides fairly good results without the resource costs of prototype implementation. [26]

In this case the memory managers, which are to be measured exist, but the test material has to be gained with a synthetic simulation. The reason for this is that there isn't any working HSDPA implementation available at the current time, which could be used to get a real allocation trace. An allocation trace is usually obtained in two different ways: by saving the events of a program when it is executed or by generating these events by hand or by some model. A hand made model combined with a simple probabilistic model is used in this work.

This chapter presents the memory management experiments done for a simulated base band DSP.

### 5.1. Description of the experiments

The complexity of UMTS/WCDMA system is much higher than in GSM or any previous generation cellular system. High complexity embedded software development with real-time requirements and many parallel tasks is a very difficult task. Therefore HSDPA software will most likely not be run on a clean hardware as in the precursor base stations and in common embedded systems. A multitasking real-time operating system (RTOS) with priority based scheduling and pre-emptive priority calls is used for handling the real-time requirements in this simulation for simulating the memory management behavior in the real HSDPA implementation.

An important aspect of memory management is the execution time. Some of the presented measurement metrics are suitable for measuring results at any point of execution when others measure the end result after the execution. Then there is the thing that the real world situation is that the memory manager will be running with the HSDPA implementation on a Node B. The boot interval of a Node B is a very long time, which is measured in months if not even years. Therefore it is the long running performance that counts the most.

Three memory managers were tested in a simulation environment with synthetic workloads. These workloads simulate the typical memory activity patterns that the HSDPA-DSP is estimated to be experiencing in a multi-user environment. The workload can be adjusted by changing the amount of simultaneous users in the simulation. Experiments with dynamic memory management are done for the most demanding part of the HSDPA DSP memory management: HARQ. It is assumed that the memory required by the flow control will be statically allocated and only the HARQ will be using dynamic memory management. Flow control was left out of the experiments because HSDPA flow control is allocating only fixed size blocks, which would enable minimum fragmentation. Only the execution time of the memory

manager would be relevant and that information is gained from these experiments done with a simulated HARQ buffer. The next section presents the memory managers that are used with the experiments.

### *5.1.1. Descriptions of the measured memory managers*

Two memory managers were used in the experiments. The memory managers have been named as RTOS pools and Block Memory Manager. Both of them are based on a different policy. Both have different properties and have a variety of optimizations. This section introduces these memory managers and their properties.

#### *5.1.1.1. RTOS pools*

In this RTOS the memory is divided into groups called “pools”. There can be several pools in one system. These pools are used to allocate signal buffers, stacks and kernel ranges. The idea of these pools is based on the segregated free lists. Each pool has a number of available block sizes that it can allocate. There is always at least one pool for system processes, which the RTOS kernel requires for its operation. In a small system all user processes might be using the same pool as the kernel. In a more complex system pools are usually created for a group of processes so that the processes and the pool are all located in the same local memory area.

The pools have their own downside. Once the pool has been created it cannot change the sizes of the block that it can allocate. This is a very serious issue if the system is a long running one like a Node B base station. Another issue is that there can only be a maximum of eight different block sizes in one pool. Now this presents a difficult problem to a dynamically allocated HARQ buffer, which is going to allocate a wide range of block sizes. The block sizes of the pool holding HARQ buffer would need to be optimized, which is a time consuming process. It is estimated that if these optimizations would be done well, it would result in a fairly good result in terms of internal fragmentation.

#### *5.1.1.2. Block Memory Manager*

Block Memory Manager (BMM) is an internal name for a developed memory manager. It is based on the segregated free lists policy. BMM is intended for dynamic memory allocations. RTOS pools are used only for interprocess signaling inside the system. BMM manages a memory area assigned to it.

The memory space is divided into master blocks. While a master block is empty it can be used for any block size. When a master block is in use only blocks, which are in a certain size range are accepted. The master blocks in use are stored with their status information in stacks for different size ranges.

When an allocation request comes in the allocator searches the stack of the desired range. If free memory cannot be found from allocated master blocks in the current stack, then a new empty master block is allocated and moved to the stack of the size. For smaller allocations than any master block is available, a sub-master block is used. The sub-master block is a fraction of the basic block size of the master block. If

a master block becomes empty when the last allocation in it is released it is then moved back to the stack of empty allocations. This reusing method makes possible to support a very wide range of different block sizes. This works only if the allocation times of the different sizes have a little spacing so that there are free master blocks available.

### 5.1.2. Synthetic workload program

For the experiments a program for creating synthetic workloads for the memory management was written with the C programming language. The workload is created with the program by using the properties of the RTOS, which supports running multiple parallel processes of the same program code. Each test process corresponds to one HSDPA user using a continuous variable bit-rate data transfer. Compiler flags control the number of user processes and the selected memory manager and so the program has to be recompiled for each manager and test case. A flow chart describing the workload program is seen in Figure 32.

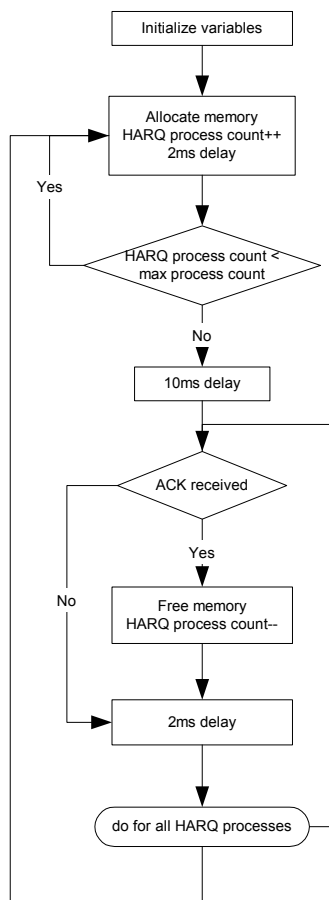


Figure 32. Flowchart of the synthetic workload program.

The program execution begins with the loading of RTOS and starting the selected number of user test processes. In the next phase the program allocates memory blocks whose sizes are defined in an array. The allocated blocks represent HARQ

processes for one user. Each allocated HARQ process means one sent Transport Block for the user. There is a 2ms delay between each allocated block. After the maximum amount of processes has been reached there is a 10ms delay. This delay represents a part of the transmission delay in a fully occupied base station, which it takes for the receiving user to respond with an ACK or NACK to the transmitted Transport Block. The next phase is to determine with a probabilistic function if the user responded with an ACK or NACK to the TB. In the case of successfully transmitted TB the allocated memory block for the HARQ process is freed. In the case of NACK the block is kept in the memory for a retransmission. Between each HARQ process there is again the 2ms delay. After all HARQ processes have been processed the program loops back to the beginning and starts again.

The RTOS switches executing processes each time when the program calls for a delay function. This way each process gets into execution.

## **5.2. Measurement metrics**

This section presents the metrics, which are used in the measurements. Measured quantities are external and internal fragmentation, performance and the maximum amount of users. The maximum amount of users indicates the number of user processes that can be supported with a specific memory amount.

### ***5.2.1. External fragmentation***

External fragmentation can be measured in several different ways. The different methods can be divided into two principal groups. In first group belong all those methods that can be calculated at a snapshot time, meaning they do not need any history information of the memory. To the second group belong all those methods that use the history information and are calculated over some period of time. This period is typically the whole runtime of the application and calculation is performed after the application has completed its execution. The second group methods therefore require that a record of the allocator activity is kept.

This section presents few different measurement methods for external fragmentation. The criteria for selecting the measurement method for the experiments done in this thesis is that it can be calculated at a snap-shot time and give results about the current situation. This is because a real base station cannot be stopped for measuring fragmentation, nor can any point be found where the application can be said to have completed its execution.

Two different measurement methods are presented. The first method is presented by Linblad in [24] and the second one is from Harrington in [27].

#### ***5.2.1.1. Linblad's method***

Linblad's method for measuring external fragmentation is simple and intuitive. It can be calculated at a snapshot time and it is fast to calculate, as it only needs two factors: the total amount of free memory and the largest free block. Equation (1) gives the Linblad's formula for calculating fragmentation.

$$f = 1 - \frac{b}{m} \quad (1)$$

where

$f$  is the fragmentation,  
 $b$  is the largest free block and  
 $m$  is the total free memory.

Fragmentation given by the equation (1) is a fraction between 0 and 1, where 1 means a system that is completely out of memory. The fragmentation of 0 is a situation where all the memory is in a single free continuous block [24]

Linblad's method is a subset of the method used by Harrington, which is described below.

### 5.2.1.2. Harrington's method

Harrington [27] introduces a method for counting external fragmentation indexes and a way to compare them. Harrington's method produces a fragmentation index, which gives higher values as fragmentation increases. In this method the ideal situation, which gives a minimum fragmentation index, is where there is only one block containing all the free memory. Harrington's method can be calculated at any time and it doesn't need any history information.

$$f = \frac{b}{l}, \quad (2)$$

where

$f$  is the fragmentation index,  
 $b$  is the breakup factor and  
 $l$  is the factor of largest free memory block.

$$b = \frac{n}{m+1}, \quad (3)$$

where

$n$  is the number of free blocks and  
 $m$  is the number of reserved blocks.

Equation (3) produces the breakup factor, which is based on the idea that the more used blocks there are, the more free blocks are potentially separating them.

$$l = \frac{s}{t}, \quad (4)$$

where

$s$  is the size of the largest free block and  
 $t$  is the total amount of free memory.

The factor of the largest free memory block is resolved from the equation (4). This equation is basically the same as Linblad's method. This describes the sizes of the fragments that the memory is broken into.

$$I = \frac{\log(a/b)}{\log 4} + 1, \quad (5)$$

where

$I$  is the relative fragmentation index,  
 $a$  is the fragmentation index to be transformed and  
 $b$  is the smallest index in the set of fragmentation indexes.

Harrington proposes the equation (5) for comparing the fragmentation indexes calculated with equation (2). This method produces comparable fragmentation indexes so that the smallest calculated fragmentation index gets 1 as its relative fragmentation index. The other larger indexes are then given values over 1, indicating how many times more fragmentation they have than the smallest one.

This method is used in the experiments. The reasons for this are that it easily gives comparable results, it is fairly simple to calculate and doesn't require any history information.

### ***5.2.2. Internal fragmentation***

Johnstone and Wilson give four methods for counting internal fragmentation in [23]. These methods count the fragmentation percentages over and above the amount of live data. All methods presented here require history information and can only be applied over a period of time. They give the increase in memory usage and not the percentage of actual memory usage that is due to fragmentation. This is because of the idea that the starting point is a perfect allocator with zero fragmentation. Johnstone and Wilson present four different methods:

1. Memory used by the allocator and divided by the requested amount by application and averaged across all points in time. This method has the drawback that it hides possible spikes in memory usage.
2. Calculated at the point of maximum live memory in use: the amount of memory used by the allocator divided by the maximum amount requested by the application. The problem is that the point of max live memory is not usually the most important point in the run of a program. An important point is likely the point where the allocator must request more memory from the OS
3. The maximum amount of memory used by allocator and divided by the memory requested by program at the point of max memory usage. The problem is that it reports high fragmentation for programs that use only a little bit more memory than they request. This happens if the extra memory is used at a point where only a minimal amount of memory is live.
4. The maximum amount of memory used by the allocator and divided by maximum amount of live memory. These are not necessarily at the same time. The problem is that this method can give a number too low if the point of maximum memory usage is a point with a small amount of live memory.

A method, similar to method 1, is used in the experiments. Internal fragmentation was counted for each allocation block size of the measured memory manager with the formula in equation (6).

$$f = \frac{a}{r}, \quad (6)$$

where

- $f$  is the internal fragmentation,
- $a$  is the allocated memory block size
- $r$  is the weighted average of requested memory.

Total internal fragmentation was gained by taking the average of the calculated internal fragmentations for each block size.

### **5.2.3. Performance**

The performance of the different memory managers is measured by comparing the clock cycles they require for the operations. From each memory manager three factors are measured from the functions used for allocation and deallocation. The measured factors are minimum, maximum and average amount of clock cycles required by the functions. The maximum amount gives the experimentally observed worst-case boundary for the execution time. The average amount of used clock cycles give the experimentally observed execution time that matters the most for a Node B kind of soft real-time system.

### **5.3. Measurement methods**

Measurement results were obtained using the TMS320C6000 Code Composer Studio (CCS) version 2.20 development environment running on a laptop computer as the tool for running the test cases. This version of CCS provides cycle accurate simulation of the Texas Instruments TMS320C6000 –series DSPs. C6000 –series include floating point, C67x generation, and fixed-point devices, C62x and C64x generations. All three use VelociTI architecture, which is a high performance VLIW architecture executing a maximum of eight 32-bit instructions per cycle.

The CCS is configured for Texas Instruments TMS320C6416 DSP with 600Mhz clock rate (cycle time 1.6 nanoseconds). A Real-Time Operating System (RTOS) is loaded first to provide the operating environment for the program executing the test cases. The RTOS can be configured to provide different memory amounts to the executed programs.

CCS provides a profiling tool which is used to count the amounts of memory references and used clock cycles in the test cases. A memory dump from the simulated DSP is saved after each executed test case. The memory dump is analyzed with an internally developed Nokia tool. Microsoft Excel and the data provided the internal tool is then used for calculating the fragmentation indexes.

#### 5.4. Test cases in the experiments

Each test case is executed with each memory manager. Table 7 shows the test cases used for external and internal fragmentation and performance experiments. The memory amount for the test cases was chosen from the Table 2 on the basis that 337kb memory should be good for finding any differences between the maximum supported user amount while measuring fragmentations.

Table 7. Test cases for fragmentation and performance

Memory (kb)	Memory (kb)
337	337
<b>Users</b>	<b>Users</b>
8	8
12	12
16	16
24	24
32	32
<b>Runtime</b>	<b>Runtime</b>
2h	12h

Table 8 shows the test cases used for testing the supported user amount with certain memory amounts. The memory amounts for the test were taken from the same Table 2 showing theoretical HARQ memory requirements.

Table 8. Test cases for user limits per memory amount

Memory (kb)	Memory (kb)
421.5	506.25
<b>Users</b>	<b>Users</b>
8	8
12	12
16	16
24	24
28	28
32	32
<b>Runtime</b>	<b>Runtime</b>
2h	2h

It was decided that 506.25 kilobytes would be the maximum limit for the tests. The amount of 421.5 kilobytes was deducted from the average of 506.25 and 337 kilobytes.

## 5.5. Results

This section and subsections show the results of the experiments described earlier. The results of the performance experiments are given first, followed by results of external and internal fragmentation. Maximum user amounts are given last.

### 5.5.1. Performance

Performance was measured for allocation and deallocation functions. Table 9 shows the results. Values are clock cycles used and categorized by the minimum, average and maximum results.

Table 9. Allocation and deallocation performances as used cycles

	Alloc			Dealloc		
	min	Avg	max	min	avg	max
BMM	184	232	324	74	95	133
Pool	92	92	100	54	54	54

When executing allocation functions BMM and pools have a larger difference than with deallocation. The results show that pools use 92 cycles on average for allocation and BMM need 232 cycles. Differences in the maximum time are even greater. BMM used 324 cycles when pools needed only 100. This gives that pools perform allocation function on average 2.5 times faster than the BMM does. With worst-case maximum used cycles the difference is 3 times more cycles for BMM than for pools. The cycle count for pools has small variations and the minimum and average are the same. Therefore the worst-case maximum 100 cycles must be a very rare.

In deallocation the BMM does better and has smaller variations than in allocation, but it still performs worse than pools. An interesting thing noted that pools have a fixed deallocation time of 54 cycles. Again the worst-case maximum deallocation time is fairly high for BMM, 133 cycles when compared with only 54 for pools. The difference is almost 2.5 times in favor of pools.

The summary is that pools perform faster than BMM. Pools are better suited for applications that require fast execution of memory management. Pools have also a very small difference between worst-case and average execution times; therefore they could perform well under hard real-time requirements. Performance results for the BMM are somewhat expected since it has a more complex algorithm. Results for the pools might change if a very large, several megabytes, memory space is used.

An interesting thing can be seen if the results are compared with the results presented by Puaut in [25]. The results are presented in Table 5 in Section 4.5. The results of Puaut and this thesis are of different magnitude, but the same characteristics are clearly visible. Sequential fits, here the pools, have a smaller average execution time and also small worst-case execution time. Segregated free lists, here the BMM, have a longer average execution time and also the big difference is notable in the worst-case time.

### 5.5.2. External fragmentation

External fragmentation was measured with two test cases. The duration of the shorter being two hours execution time and the longer was 12 hours. First are the results of the shorter run.

Figure 33 shows the results of external fragmentation in the 2 hour test cases. In the x-axis is the user count and y-axis is for the values of the relative fragmentation index.

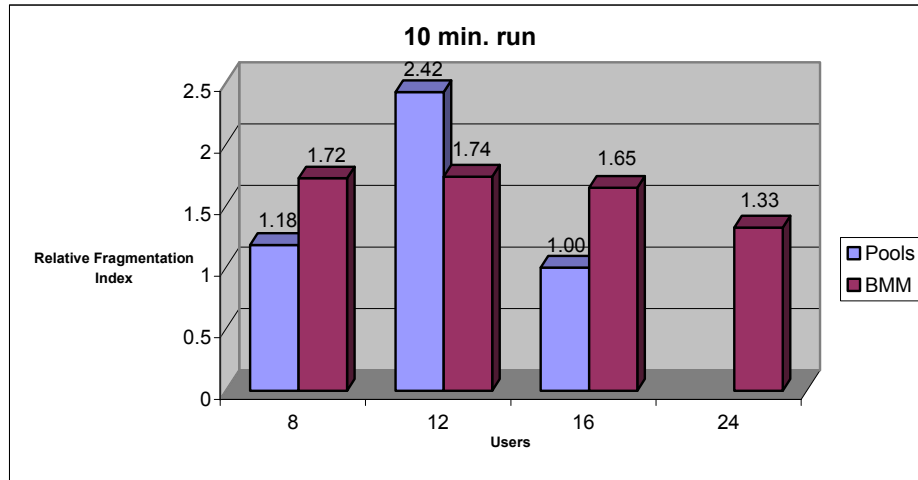


Figure 33. External fragmentation in 10 minute execution.

From the figure can be seen that BMM seems to have about the same relative fragmentation index count with all user counts. The fragmentation index for pools has high variations, but is generally smaller than BMM. Pools feature an exceptionally low fragmentation index with 16 users. Pools could not support 24 users and ran out of memory. The same happened with an experimental amount of 20 users. BMM supported 24 users, but ran out of memory with 32 users.

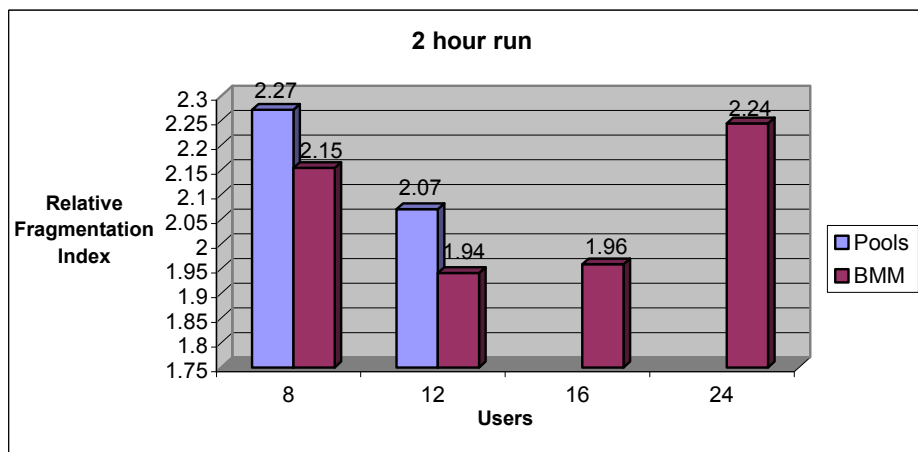


Figure 34. External fragmentation in 2 hour execution.

In Figure 34 the external fragmentation results is seen from the longer 2h execution. The most important discovery is that pools ran out of memory with 16 users in this longer run. In the 10 minute run 16 users executed well with pools. This a classical example of fragmentation that slowly advances and finally the system runs out of memory.

Pools recorded the highest relative fragmentation value of over 2.25 with eight users. The fragmentation index for BMM is quite different than in the 10 minute run. BMM resulted in quite high fragmentation index with 24 user load. Generally the fragmentation index seems to be at its minimum with an intermediate load of 12 to 16 users. BMM could support 24 users and pools only 12. The fragmentation of the pools decreases when the user count increases. This could be explained by the fact that, with less users, there are more possibilities to fragment the memory.

### 5.5.3. Internal fragmentation

The results for the internal fragmentation experiments are seen in Figure 35. Both memory managers produce some internal fragmentation, as expected.

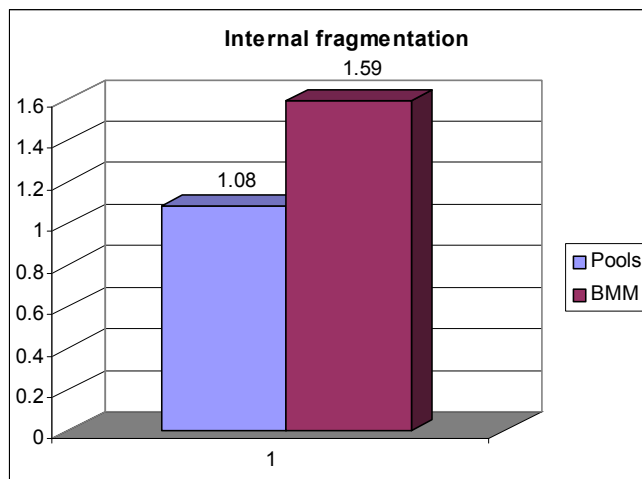


Figure 35. Internal fragmentation.

BMM has a much higher amount of internal fragmentation than pools. The memory managed by BMM is internally fragmented almost 1.5 times as much as the one managed with pools. This is direct result of the segregated free lists and good-fit policy. The spacing between the list sizes is too big and this is the reason for the results seen in Figure 35. Pools perform quite well in terms of internal fragmentation. Pool sizes were hand optimized for the experiment and BMM creates the sizes automatically. This shows that hand optimized pool sizes are very effective for decreasing internal fragmentation when the allocation pattern and sizes are known.

The summary is that BMM loses quite a bit of its memory capacity to the internal fragmentation. If this arises for a significant problem pools are a better choice than BMM. Especially if HARQ would get a dedicated pool with hand optimized block sizes.

The gained results are quite similar to those presented by Johnstone and Wilson in [23] and also presented in a compacted form in Table 6 in Section 4.5. Similar results

are that the sequential fits feature very low internal fragmentation and segregated free lists, on which the BMM is based, have a higher count. Though the results of Lea's allocator in Table 6 show that it is possible to create a segregated free lists allocator that doesn't feature heavy internal fragmentation.

#### 5.5.4. Maximum User Amounts

The experiments to find out maximum user amounts with different amounts of memory were conducted after the fragmentation experiments. Fragmentation experiments revealed that with 337 kilobytes only BMM supported 24 users. The maximum user amount results from fragmentation experiments are presented in Table 10 with the results from the maximum user amount experiments. Table 10 shows that when the memory is increased to 421.5 kilobytes pools can support 16 users, but still not 24.

When the amount was increased to 506.25 kilobytes it was expected that the pools could achieve the 32-user limit. Unfortunately for some reason the test environment became unstable and execution was unable to finish in each attempt. However, BMM ran the tests successfully with also the higher 32 user memory amount.

Table 10. Maximum users

	377 kB		421.5 kB		506.25 kB	
8	Ok	Ok	Ok	Ok	Ok	Ok
12	Ok	Ok	Ok	Ok	Ok	Ok
16	mem. out	Ok	Ok	Ok	Ok	Ok
24	mem. out	Ok	mem. out	Ok	Ok	Ok
28	mem. out	mem. out	mem. out	Ok	Unstable	Ok
32	mem. out	mem. out	mem. out	Ok	Unstable	Ok
	<b>Pools</b>	<b>BMM</b>	<b>Pools</b>	<b>BMM</b>	<b>Pools</b>	<b>BMM</b>

The summary of these test cases is that the BMM supports more users with a smaller amount of memory. This may seem a bit strange because BMM had a much higher internal fragmentation count. The reason for the success of the BMM is its efficient allocation and deallocation policy, which makes possible a very good reusability of any released blocks.

## 5.6. Summary of the results

From the results it can be seen that pools are a simple and effective method for handling dynamic memory management with DSP. This is not very surprising since they are implemented in a highly successful commercial RTOS. They have their limitations though. In these experiments even the optimized pools could not support a very high amount of users. This is the downside of their simplicity that gave such good results in the performance tests. The external fragmentation of the pools can

have large variations as the results show. Therefore it is recommended to keep the fragmentation in mind if working with pools.

BMM is slightly slower to execute and this is a clear result of its more complex operation. BMM has also quite large variations in its execution time, which should be taken into account if tight scheduling is required. The main advantage of BMM is its ability to make efficient use of the memory space assigned to it. It has some external fragmentation and a little more internal fragmentation than pools, but its efficient reusing ability makes possible to support more simultaneous users.

The results gained from these experiments support the results of previous publications. The performance results gained were in line with those presented by Puaut and fragmentation results corresponded to those presented by Johnstone and Wilson.

The large difference in the amount of maximum supported users is severe, if considering the use of pools for a application like HARQ buffers. The results are such that if there is an alternative to the pools it should probably be chosen. Of course only if the alternative manager promises better performance.

Both measured memory managers are good in their own way and if a choice must be made, it must be based on the current requirements. If memory space is scarce then the BMM is the winner, but if execution time is the one that matters the most then pools should be chosen. Though, the BMM has very low CPU cycle requirements. Both, pools and BMM, are suited for use with a DSP. There are many other memory managers and policies, which were not used in these experiments, would probably require much more cycles. Possibly they could decrease the fragmentation levels, but usually with high cost of cycles. Choosing the right memory manager is always a matter of compromises. It should be decided, which feature is the most critical and most often used. The memory manager should then be optimized for this feature as much as possible without exceeding the budget in other areas.

## **6. DISCUSSION AND FUTURE DEVELOPMENTS**

### **6.1. Reaching the goals of the thesis**

The goal of this thesis was to study methods for measuring the memory management, measure the performance of two memory managers used with a DSP and to perform an evaluation of these memory managers from the viewpoint of a DSP in HSDPA base station.

There are several previous publications existing that deal with external and internal fragmentation. The methods for the measurements done in this thesis were chosen from previous publications and with a little fine-tuning they were put to use in the experiments. There is no general right way for measuring memory management. The criteria were that both should be able to be calculated at a snapshot time. A method published by Harrington met the requirement and was chosen for external fragmentation. For internal fragmentation a method published by Johnstone and Wilson was chosen. The clock cycles used by the memory managers were measured by using a profiling tool of the CCS. The memory manager's ability to utilize the assigned memory was measured by incrementally increasing load for each total memory amount and then monitoring an error code indicating insufficient memory.

In the experiments four different factors were measured: performance in terms of used clock cycles, internal and external fragmentation and the maximum amount of HARQ buffered HSDPA users with different amounts of memory. The experiments were done with a simulated DSP. The simulator is produced by the manufacturer of the DSP and is cycle-accurate. Therefore the results should be exactly the same with real hardware. The measured memory managers were evaluated through the results of the experiments.

This work will be used as a guideline and basic research material in the development process of third generation base stations. It gives actual documented material of the performance of the internally developed Block Memory Manager and of the RTOS pools. This information can be used to further develop the BMM if necessary. Also any new improvements of the BMM can be measured with the methods and against the results gained in this thesis.

There are several publications available that are dealing with memory management and the efficiency of different allocators. Very few of those study long-lived processes. No single study of DSP memory management was found during the work of this thesis. Therefore the validity of this thesis is considered good as it presents the well-studied field of memory management from a completely new viewpoint.

### **6.2. Enhancing the experiments**

The experiments performed in this thesis could be enhanced in a couple of ways. The first way is to use more memory managers that use different allocation policies. This would provide a wider view of the actual performance properties of different allocation policies. This information of costs and benefits of the different policies could help greatly in any future design decisions.

Another improvement aspect would be running similar experiments using a real allocation trace from actual HSDPA traffic. This would of course require a working HSDPA base station. A real allocation trace would give a more accurate view of the performance of memory manager than the synthetic one used in the experiments made in this thesis.

Also longer execution times could be used, because usually fragmentation tends to increase with time. To get most of the results these improvements could be combined so that memory management would be measured with a real allocation trace and the experiment would be running for several days per test case. This way the true information of the long running process behavior could be gained.

### **6.3. Ideas for developing DSP memory management**

Some ideas for developing DSP memory management occurred during the work. Segregated free lists could benefit from using dynamic block sizes to minimize internal fragmentation. This could present high costs in execution time since dynamic block sizes would require quite complex control operations.

In DSP applications the advantages gained from hand optimizing the block sizes should not be overlooked. This can greatly reduce the effects of internal fragmentation.

If external fragmentation becomes a problem then compaction with handles could provide a solution. In a base station memory management compaction could be done in a deferred mode and initiated with a trigger. The trigger could be set for some time at night when there typically is low activity. The compaction process could be executed without interfering with the operation of the base station.

## 7. CONCLUSIONS

This thesis has studied DSP memory management. The characteristics of a high-speed data transfer in the form of HSDPA give the background and framework for the studies. In the study of HSDPA it was discovered that for the memory management the most challenging parts are flow control from RNC to the Node B and the HARQ flow control from Node B to the UE. The reasons are the large maximum amounts of required memory, wide variation in allocation patterns and block sizes and tight time limits.

The differences of a TI C67x DSP, PowerPC -processor targeted for embedded applications and a desktop PC-processor Athlon were studied. It was found that the embedded PowerPC is capable of executing heavily branching memory management code with over 100% speed advantage compared to the C67x DSP. This was found out to be a direct result of the limited hardware support for branch resolution and prediction. The difference between desktop the Athlon and the C67x is even greater. C67x is very slow at executing branching code.

Studies of memory management methods showed that there are several different methods available. Each method has its own drawbacks. The method used with DSP should be

- fast to execute,
- deterministic and
- feature minimum fragmentation.

Methods using the sequential fit policy are simple but usually have a large execution time variation, heavy external fragmentation and bad scalability to large memory spaces. Segregated free lists have the drawback of internal fragmentation if the list size spacing is set too wide compared to the allocation sizes. It is possible to implement a segregated free lists allocator that doesn't feature heavy internal fragmentation by increasing the amount of list sizes. Segregated free lists have theoretically lower search times than the sequential fits, but results from the experiments of this thesis and previous publications showed that in practice they are slower than sequential fits. Segregated lists scale better to large memory spaces.

The studies of general memory management problems showed that for DSP memory management the most dangerous problems are the internal and external fragmentation.

In the experiments it was found that the pools and BMM have great differences in performance. Pools were found to be fast and simple to execute and performed well with small to intermediate loads. When the number of users in the synthetic workload program was increased, the pools quickly ran out of memory. BMM could support a much higher number of users with the same amount of memory. The reason for this is its good reusing method. The large difference between the pools and BMM in memory utilization with a high load is a significant discovery in the favor of BMM. This practically rules out the use of pools in a situation similar to the experiments.

The results of the experiments can be summarized into three facts:

- CPU cycle consumption is within acceptable limits with both of the measured managers. Both of them are suited for use with a DSP. There is no doubt that many other memory managers would use unacceptably high

amount of cycles and would be unsuited for use in a DSP, but do fine in general-purpose processors.

- Fragmentation is the critical distinctive feature in the measured managers. BMM kept its fragmentation levels in control and it didn't hamper the whole system.
- As a whole the BMM is better choice for memory management, because it can make more use of the memory space assigned to it without overstressing the CPU. Pools require more memory than the BMM to be able to match the performance of BMM.

If the use of either the pools or BMM is considered, their differences should be taken into account. If some other memory manager is used its advantages and drawbacks should be carefully considered.

## 8. REFERENCES

- [1] 3GPP Technical Report 21.01 V3.0.1, Universal Mobile Telecommunications System; Requirements for the UMTS Terrestrial Radio Access system (UTRA) (1997). The 3<sup>rd</sup> Generation Partnership Project, 25 p.
- [2] Overview of 3GPP Release 5, Summary of all Release 5 Features (2003). ETSI Mobile Competence Centre, 44 p.
- [3] Holma H. & Toskala A. (2001) WCDMA for UMTS Radio Access For Third Generation Mobile Communications. John Wiley & Sons, Ltd, West Sussex, England, 313 p.
- [4] 3G System Training (2001). Nokia company confidential training material. 352 p.
- [5] Nokia internal document. Document code B6O S6013F9E
- [6] Qiu R., Zhu W & Zhang Y. (2002) Third-Generation and Beyond (3.5G) Wireless Networks and Its Applications. In: IEEE International Symposium on Circuits and Systems, May 26 – 29, Scottsdale, United States of America, Vol. 1, p. I-41 - I-44.
- [7] Parkvall S., Dahlman E., Frenger P., Beming P & Persson M. (2001) The high speed packet data evolution of WCDMA. In: 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, September 30 – October 3, Vol. 2, p. G-27 - G-31.
- [8] 3GPP Technical specification 25.321 V6.0.0, Technical Specification Group Radio Access Network (2003). Medium Access Control (MAC) protocol specification (Release 6), The 3<sup>rd</sup> Generation Partnership Project, 61 p.
- [9] 3GPP Technical Report 25.858 V5.0.0, Technical Specification Group Radio Access Network (2002). High Speed Downlink Packet Access: Physical Layer Aspects (Release 5), The 3<sup>rd</sup> Generation Partnership Project, 31 p.
- [10] Nokia internal document. Document code DN03406865.
- [11] Kolding T., Pedersen K., Wigard J., Fredriksen F. & Mogensen P. (2003) High Speed Downlink Packet Access: WCDMA Evolution. IEEE Vehicular Technology Society (VTS) News, vol. 50, no.1, p. 4-10.
- [12] 3GPP Technical report 3G 25.848 Version 4.0.0 (2001). Physical layer aspects of UTRA High Speed Downlink Packet Access, The 3<sup>rd</sup> Generation Partnership Project, 89 p.

- [13] Kolding T., Frederiksen F. & Mogensen P. (31.3.2004) Performance Aspects of WCDMA Systems with High Speed Downlink Packet Access (HSDPA). URL: [http://nds2.ir.nokia.com/downloads/aboutnokia/research/library/mobile\\_networks/MNW22.pdf](http://nds2.ir.nokia.com/downloads/aboutnokia/research/library/mobile_networks/MNW22.pdf).
- [14] Parhi K. & Nishitani T. (1999) Digital Signal Processing for Multimedia Systems. Marcel Dekker Inc., New York, 855 p.
- [15] Silvén O. (2002) Lecture material for Signal Processing Systems. University of Oulu, 74 p.
- [16] TMS320C6000 CPU and Instruction Set Reference Guide (14.3.2004). Literature Number: SPRU189F. Texas Instruments, 685 p.
- [17] Motorola G2 PowerPC Core Reference Manual (14.4.2004). URL: [http://www.motorola.com/files/32bit/doc/ref\\_manual/G2CORERM.pdf](http://www.motorola.com/files/32bit/doc/ref_manual/G2CORERM.pdf).
- [18] AMD Athlon™ Processor x86 Code Optimization Guide (14.3.2004). Publication no. 22007, revision K. URL: [http://www.amd.com/us-en/assets/content\\_type/white\\_papers\\_and\\_tech\\_docs/22007.pdf](http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22007.pdf).
- [19] Hennessy J. & Patterson D. (2002) Computer Architecture: A Quantitative Approach, Third Edition. Morgan Kaufmann Publishers, 1136 p.
- [20] The Memory Management Reference (23.2.2004). URL: <http://www.memorymanagement.org>.
- [21] Silberschatz A., Peterson J.L. & Galvin P. (1998) Operating System Concepts, 5<sup>th</sup> edition, Addison-Wesley Publishing Company. 888 p.
- [22] Wilson R, Johnstone M, Neely M & Boles D (1995). Dynamic Storage Allocation: A Survey and Critical Review. In: International Workshop on Memory Management, September 27 - 29, Kinross, Scotland, UK, URL: <ftp://ftp.cs.utexas.edu/pub/garbage/allocsrv.ps>, 30.12.2003. 78 p.
- [23] Wilson P. & Johnstone M. (1998) The Memory Fragmentation Problem: Solved? In: Proceedings of the first international symposium on Memory management, October 17 – 19, Vancouver, British Columbia, Canada, p. 26 – 36.
- [24] Lindblad J. (2001) Memory Fragmentation Guide. OSE Systems OSE Guide. 13 p.
- [25] Puaut I. (2002) Real-Time Performance of Dynamic Memory Allocation Algorithms. Publication interne No 1429, Institut de Recherche en Informatique et Systèmes Aléatoires, Campus Universitaire de Beaulieu, Rennes Cedex, France. (25.3.2004) URL: <http://www.irisa.fr/slidor/doc/ps02/ecrts02.ps.gz>.

- [26] Zorn B. & Grunwald D. (1994) Evaluating Models of Memory Allocation. ACM Transactions on Modeling and Computer Simulation, Vol. 4, No. 1, p. 107-131.
- [27] Harrington J. (1993) Measuring fragmentation. Dr. Dobb's journal, April 1993, p. 1-16.

## 9. LIST OF FIGURES

Figure 1. Example of base station code resource division.....	9
Figure 2. Differences of typical signal processing and memory management sequences.....	10
Figure 3. Conceptual UMTS network architecture.....	12
Figure 4. Network elements of a UMTS network.....	13
Figure 5. UTRAN Channel types and their locations.....	14
Figure 6. Logical to transport to physical channel mapping in downlink direction...	15
Figure 7. Logical to transport to physical channel mapping in uplink direction.....	15
Figure 8. HSDPA Features.....	17
Figure 9. HSDPA Related channels (for one user).....	18
Figure 10. UTRAN MAC architecture.....	19
Figure 11. MAC-d PDU structure.....	20
Figure 12. MAC-hs detailed architecture.....	20
Figure 13. MAC-hs PDU.....	21
Figure 14. Flow control principle.....	22
Figure 15. Proportional Fair Resource and the top of the fades.....	24
Figure 16. Principle of Chase Combining.....	25
Figure 17. Principle of Incremental Redundancy.....	25
Figure 18. Examples of HSDPA data rates with QPSK and 16QAM modulations...	28
Figure 19. HSDPA Operation concept.....	29
Figure 20. Typical DSP Environment.....	32
Figure 21 General block diagram of WCDMA Node B architecture.....	32
Figure 22. Block diagram of HSDPA DSP environment.....	33
Figure 23. TMS320C67x pipeline.....	33
Figure 24. Internal fragmentation.....	40
Figure 25. External fragmentation.....	41
Figure 26. Coalescing blocks of memory.....	42
Figure 27. Splitting a block of memory.....	43
Figure 28. Boundary tags and their optimization.....	43
Figure 29. Double-linked list of Sequential Fits.....	44
Figure 30. Segregated Free Lists.....	45
Figure 31. Principle of Buddy systems.....	46
Figure 32. Flowchart of the synthetic workload program.....	51
Figure 33. External fragmentation in 10 minute execution.....	58
Figure 34. External fragmentation in 2 hour execution.....	58
Figure 35. Internal fragmentation.....	59
Table 1. Estimations of Flow Control memory requirements.....	22
Table 2. HARQ memory requirements.....	26
Table 3. Processor comparision.....	36
Table 4. Branch execution times in clock cycles.....	37
Table 5. Puaut's table of allocation performance.....	47
Table 6. Fragmentation results.....	48
Table 7. Test cases for fragmentation and performance.....	56
Table 8. Test cases for user limits per memory amount.....	56
Table 9. Allocation and deallocation performances as used cycles.....	57
Table 10. Maximum users.....	60

**APPENDICES**

Appendix 1

Transport Block Sizes

## APPENDIX 1 TRANSPORT BLOCK SIZES

<b>Index</b>	<b>TB Size</b>	<b>Index</b>	<b>TB Size</b>	<b>Index</b>	<b>TB Size</b>
1	137	86	1380	171	6324
2	149	87	1405	172	6438
3	161	88	1430	173	6554
4	173	89	1456	174	6673
5	185	90	1483	175	6793
6	197	91	1509	176	6916
7	209	92	1537	177	7041
8	221	93	1564	178	7168
9	233	94	1593	179	7298
10	245	95	1621	180	7430
11	257	96	1651	181	7564
12	269	97	1681	182	7700
13	281	98	1711	183	7840
14	293	99	1742	184	7981
15	305	100	1773	185	8125
16	317	101	1805	186	8272
17	329	102	1838	187	8422
18	341	103	1871	188	8574
19	353	104	1905	189	8729
20	365	105	1939	190	8886
21	377	106	1974	191	9047
22	389	107	2010	192	9210
23	401	108	2046	193	9377
24	413	109	2083	194	9546
25	425	110	2121	195	9719
26	437	111	2159	196	9894
27	449	112	2198	197	10073
28	461	113	2238	198	10255
29	473	114	2279	199	10440
30	485	115	2320	200	10629
31	497	116	2362	201	10821
32	509	117	2404	202	11017
33	521	118	2448	203	11216
34	533	119	2492	204	11418
35	545	120	2537	205	11625
36	557	121	2583	206	11835
37	569	122	2630	207	12048
38	581	123	2677	208	12266
39	593	124	2726	209	12488
40	605	125	2775	210	12713
41	616	126	2825	211	12943
42	627	127	2876	212	13177
43	639	128	2928	213	13415

44	650	129	2981	214	13657
45	662	130	3035	215	13904
46	674	131	3090	216	14155
47	686	132	3145	217	14411
48	699	133	3202	218	14671
49	711	134	3260	219	14936
50	724	135	3319	220	15206
51	737	136	3379	221	15481
52	751	137	3440	222	15761
53	764	138	3502	223	16045
54	778	139	3565	224	16335
55	792	140	3630	225	16630
56	806	141	3695	226	16931
57	821	142	3762	227	17237
58	836	143	3830	228	17548
59	851	144	3899	229	17865
60	866	145	3970	230	18188
61	882	146	4042	231	18517
62	898	147	4115	232	18851
63	914	148	4189	233	19192
64	931	149	4265	234	19538
65	947	150	4342	235	19891
66	964	151	4420	236	20251
67	982	152	4500	237	20617
68	1000	153	4581	238	20989
69	1018	154	4664	239	21368
70	1036	155	4748	240	21754
71	1055	156	4834	241	22147
72	1074	157	4921	242	22548
73	1093	158	5010	243	22955
74	1113	159	5101	244	23370
75	1133	160	5193	245	23792
76	1154	161	5287	246	24222
77	1175	162	5382	247	24659
78	1196	163	5480	248	25105
79	1217	164	5579	249	25558
80	1239	165	5680	250	26020
81	1262	166	5782	251	26490
82	1285	167	5887	252	26969
83	1308	168	5993	253	27456
84	1331	169	6101	254	27952
85	1356	170	6211		