CELP based speech compression a VLSI implementation

Master thesis
Stefan Lundberg, E-92
Sven Karlsson, F-92

1996–12–18

Abstract

This thesis describes a mixed mode ASIC implementation of a CELP CODEC called Darwin. Darwin can code CELP in real-time at a clock frequency of 14 MHz. Darwin consists among other things of a DSP unit and a RISC processor both developed and implemented during this thesis using VHDL. To the DSP unit the CELP algorithm has been successfully mapped. The DSP unit is also capable of speeding up CELP's stochastic code book search with a factor of 6 yielding an overall speedup of 60%. Darwin is implemented using a 0.8μ CMOS process. Its core size is 30 mm^2 and it can be clocked at 50 MHz.

Supervisors:
Pietro Andreani
Johan Petersen
Jörgen Olsson

Foreword

This thesis is a compulsory part of the educations in Electrical Engineering and Engineering Physics at the Institute of Technology, Lund University. The work was done at the Department of Applied Electronics during the spring, summer and autumn of 1996.

We are grateful to many people for constructive comments and suggestions, and for their help in improving our design. The authors would like to send special thanks to our supervisors and the following persons. Without their invaluable support and good ideas this thesis would not have been possible:
Björn Briedegard and Eric Jonsson.

Stefan would like to send a special thank to Kristina for being ever so patient. Sven thanks Madelen for putting up with the never ending succession of late evenings and early mornings.

Lund, December 1996

Sven Karlsson

Stefan Lundberg

Contents

1 Introduction

CELP is a well known speech compression standard with very good sound quality at a low bitrate. The major disadvantage however is that it is very compute intensive and thus it is not used much. This thesis describes a VLSI implementation of a CELP coder and decoder running at a moderate clock frequency.

Chapter 2 describes the background of the thesis while Chapter 3 describes CELP and chapter 4 discusses the used design process. Chapter 5 shows the major design trade-offs. The implementation is described with more details in chapter 6, analog, and 7, digital. Conclusions and summaries can be found in chapter 8. The appendix describes the CPU instruction set, speech coders in general and calculations concerning the analog design.

2 Background

Every product containing electronics manufactured and sold today must be improved to attract the future buyer. Tomorrow, the market demands a product which is smaller, cheaper, less power consuming and has higher functionality. Key components in those systems are custom developed integrated circuits manufactured on silicon in large series at a very low price.

To all basic requirements on the hardware adds the fact that the amount of software in an embedded system tends to grow every year thereby moving the main development costs from the hardware into the software part of the product. This introduces new factors into the specification, when hardware for a new product is planned. The hardware designer must early analyze how proposed hardware affects the software in both the planned product and a future improved implementation. The next generation of the system must probably reuse large portions of the software to cut time and cost.

Special processors and custom developed micro electronics are very important for functionality and performance of future hardware, and it is important to build up knowledge and skill in this fast moving field to be able to develop successful products.

This project was done in close cooperation between Business Security AB, the Department of Applied Electronics and the Department of Computer Engineering LTH, Lund.

2.1 Business Security AB

Business Security AB is a small company (20 employees in 1996) in Lund developing and producing products for hardware encryption of data, voice and fax communication. Today the following categories of products exist:

- SecuriCrypto, for secure data communication.

- SecuriFax, for secure fax transmission.
- SecuriVoice, for secure phone lines.
- KeyBase, for key management.

All products contains standard components like microprocessors, DSP:s, FPGA:s etc. To increase the technical and knowledge level they have been looking for a way to integrate some of the functions of their products into Application Specific Integrated Circuits (ASIC:s). This would have a number of positive effects:

- Improved functionality
- Safer crypto
- Harder to copy
- Physically smaller
- Less power consumption
- No export limitations

This could make the volume larger, helping to cut cost from the product and thereby resulting in better profit for each sold unit.

2.2 Higher education cooperation

To make it possible for a small company to start developing their own ASIC:s, close cooperation with education centers is quite important. The first step towards this type of development tends to be very difficult but cooperation may minimize the step and move necessary knowledge into the company.

Since this type of development includes quite high initial cost, long development and unfortunately some risks, it is extremely valuable for the company to get a good start with right equipment, tools, working routines and contacts.

This time the cooperation was arranged with LTH and the Department of Applied Electronics with the help of some tools and methods from The Department of Computer Science.

2.3 Department of Applied Electronics

The Department of Applied Electronics is working in the field of ASIC implementation of custom DSP's as well as analog hf-amplifiers. The DSP research is focused at bit serial implementation and automatic tools to generate datapath and control. Target applications are future wide band systems for mobile communications.

The resources we used at the department were different SUN workstations running the ASIC design software Cadence [Zej95]. Cadence is a quite large package of programs with a large palette of tools from which we have used the following: Composer, VHDL toolbox, Synergy, Leapfrog XL, AnalogArtist with spectreS and spectraSVerilog, BlockEncemble, CellEncemble, Layout editor, DRS and LVS.

2.4 Department of Computer Engineering

The Department of Computer Engineering has developed the VHDL tool BBDS [BBDS] that was mainly used for the development of the digital parts of the chip. The idea of Werner diagrams [Werner] is used to produce a clock cycle true model of the design in a short period of time. BBDS helps the designer to implement error free pipelined designs with help of an integrated simulator which has very short turnaround time. For more information about how we used this tool see section 4.3.

2.5 SecuriVoice

Figure 1: SecuriVoice

The current product for voice crypto is called SecuriVoice, see figure 1, which is a box placed under the telephone, connected between the telephone unit and the

outgoing telephone line. The cryptokey is stored on a smart card (Chip-card) and if both ends of the communication line have SecuriVoice and the same key, crypto mode may be entered and safe voice communication can be ensured.

Technically the function of the box is as follows:

The recorded talk signal is speech coded with a Texas DSP running the CELP-algorithm forwarded to the crypto unit and transmitted over the public telephone network at 9600 bps. After modem transportation the data is decrypted, speech is restored by the CELP-algorithm and played back to the user. To make sound recording and sound playback possible an analog interface, CODEC, containing A/D- and D/A-converters is used.

2.6 Defining the thesis

During this thesis an ASIC, later called Darwin, containing the key functions of the current product SecuriVoice should be developed. Figure 2 shows the system divided into 4 blocks, namely crypto, CELP, modem and control. In this thesis the CELP and control block should be implemented, leaving 2 blocks Crypto and Modem for later projects.

The design should be prepared for an on-chip crypto unit and an external modem circuit.

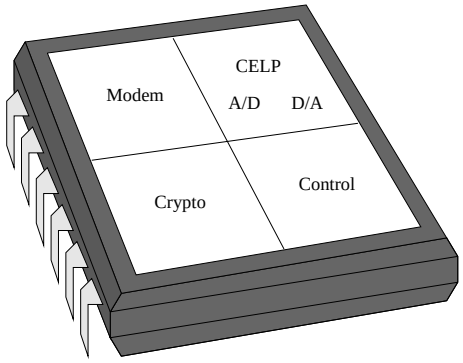


Figure 2: The first sketch

The ASIC should be developed for a 0.8 μm CMOS process, AMS cyx [CYX], using a standard cell library for digital blocks and full custom layout for analog blocks. The chip area should be small, the number of pins few and the process is a simple digital process with no extra polysilicon layers or laser trimmed components. The analog interface must be integrated together with the digital part into the same chip and the design should be open for the future and the digital part must be written in the hardware describing language VHDL.

2.7 Specification

The ASIC must of course be small, about 40 mm^2, and fit in a reasonable package less than 160 pins. Special effort should be put into minimizing the usage of external RAM and ROM.

EMI should be considered by minimizing clock frequency and choosing driving requirements for all pads depending on the individual signal.

- Global chip
  - 3.3 V supply (±5 %)
  - 144 PQFP package with 25 ^{o} C / W
  - Ambient, max 50 ^{o} C
  - Die, max 85 ^{o} C
  - Power consumption 1.8 W
- Analog blocks, CODEC
  - 2 channels ADC 16 kHz 10 bit
  - 2 channels DAC 16 kHz 10 bit
  - Separate 3.3 V (±2.5%)
  - External voltage reference
  - Input range from 0.5 V to 2.5 V
- Digital blocks
  - Some type of control CPU with program in external ROM.
  - Possible to interface with an on chip crypto block
  - Support for the CELP algorithm in hardware
  - Shared RAM for CELP hardware, crypto and CPU.
  - General I/O ports
  - UART

3 Introduction to CELP speech coding

The following section describe the CELP speech coding algorithm with focus on implementional requirements and problems. In appendix B an introduction to speech coding can be found which can be recommended to read before this secion if speech coding is an unknown field to the reader.

3.1 CELP coder

CELP is an analysis-by-synthesis frame-oriented speech coder with two perceptually weighted vector quantizations and one linear predictor filter. The speech is sampled with 12 bits at 8 kHz and split into 30 ms frames which are analyzed as separate units. The linear predictor filter is a 10th order all pole filter used to remove the short term correlation spectrum from the speech, ie a Linear predictive coder, LPC, analysis. An adaptive pitch code book is used to remove long time signal periodicity, ie the speakers pitch. The stochastic code book is then used to vector-quantisize the residual to save information in the speech signal not detected by earlier stages. All code book indexes and gain values are chosen to minimize the perceptually weighted distortion measure. The perceptually weighting function uses the masking properties of human hearing to improve the subjective speech quality. The speech is compressed by a factor 20:1. Encoding requires at least 10 times more calculations than decoding.

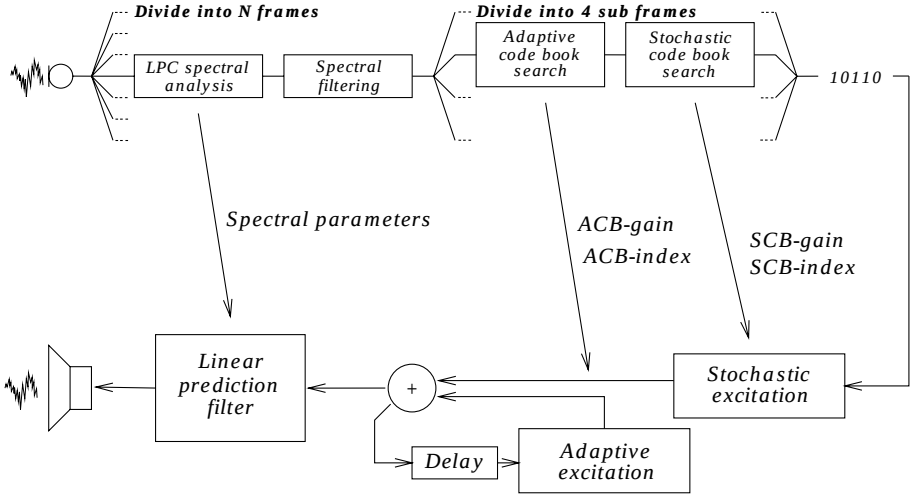


Figure 3: Overview of CELP-algorithm

3.1.1 Code book searches

The code book searches is the hard part of the CELP algorithm. During the search all vectors stored in the code book are evaluated as a candidate to be

the best matching index. Once found, only the index from the code book and correct gain needs to be transmitted in order to reconstruct the speech. To reduce complexity for code book searches, the frame is divided into 7.5 ms sub frames on which all calculations and searches are done.

3.1.2 The stochastic code book

The stochastic code book, SCB, contains sparse overlapping ternary valued pseudo randomly generated codewords with a 2 samples overlap. In other words, the SCB is a single long vector with values -1, 0, 1 from which 512 different vectors with length 60 can be extracted by moving a pointer 2 steps forward for every index.

3.1.3 The adaptive code book

The adaptive code book, ACB, or pitch code book is overlapped with shift 1 and contains a history of past excitation signals. The pitch code book includes 128 integer delays and 128 non integer delays which gives 256 different pitch frequencies ranging from 54 - 400 Hz. The index is the number of samples back in time where the best new excitation is found.

3.2 The CELP analyzer

The analyzer contains a tunable synthesizer, a weighting filter and an algorithm tuning the synthesizer to minimize the difference between the original and the generated speech, see figure 4. The analyzer process is a closed-loop minimizing the perceptual weighted error. The LPC filter is found by an open-loop spectrum analysis of the input signal. The analyzer is the most compute intensive part of CELP.

3.2.1 Analysis process

The spectral information from the PCM-sampled speech is first obtained by a LPC-analysis resulting in 10 filter parameters which are converted to a more efficient Line Spectrum Pair, LSP, notation, see figure 3. The frame of speech is then divided into 4 sub frames and the perceptual weighting filter for each sub frame is interpolated from the LSP parameters.

For each sub frame the weighting filter impulse response is calculated and used as input to the adaptive and stochastic code book searches. The adaptive code book search finds the pitch in the sound which is removed and the stochastic code book search looks for the best noise like excitation signal of the residual. The compressed speech which is sent to the receiver, contains a set of LSP parameters, pitch parameters (index and gain) and stochastic code book parameters (index and gain).

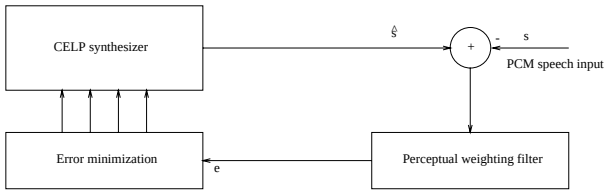


Figure 4: CELP analyzer

3.3 The CELP synthesizer

The synthesizer is a two stage code excitation of a LPC filter. Stochastic noise is picked from the code book, amplified and then added to the output from the adaptive pitch code book, see figure 5.

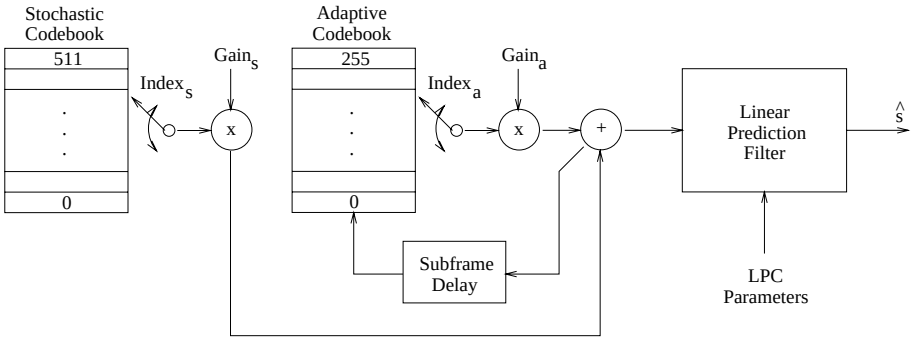


Figure 5: CELP synthesizer

3.3.1 Synthesis process

The synthesizer unpacks all bits and does some error recovery before the speech is restored. First LSP parameters are used to reconstruct the perceptual weighting filter. Then the vector pointed to by the stochastic code book index is fed into a variable length tap-filter controlled by the pitch index thereby synthesizing the speech. This speech is then used as excitation of the weighting filter to reconstruct the PCM-coded speech.

3.4 US Federal standard 1016

The CELP coder has been made a US Federal standard [Std1016] by the Department of Defense,DoD [CaTW90]. This standard includes CELP speech parameters, bit allocation scheme and Hamming error protection coding. Every CELP frame is packed into 144 bits with a bit rate of 4.8 kb/s. The standard

does not specify how to obtain the compressed speech, thus different algorithms may be used.

From an inspection of the standard document it is almost impossible to understand how CELP works and even harder to find out how it can be implemented. DoD has as an additional source of information released an example implementation of CELP available in Fortran as well as C. This implementation has been our main source of information and has been used in our software models.

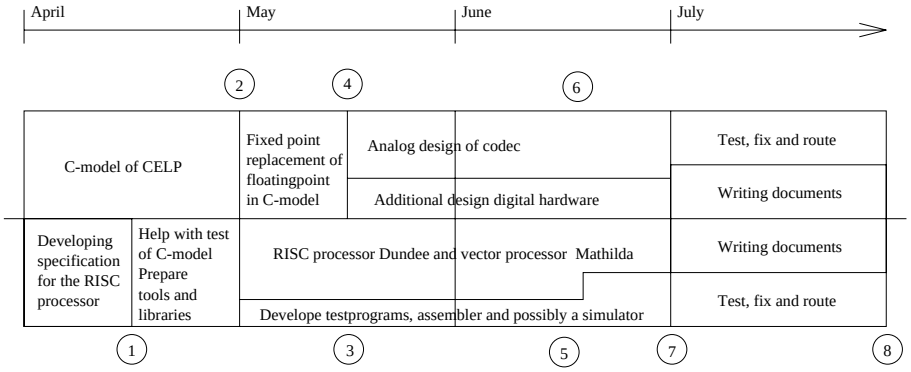
3.5 CELP algorithm

The implementation of real time CELP analysis is, due to its high calculation requirement and its complexity, far from easy. A naive implementation with exhaustive search in both code books is so compute intensive that real time compression is out of the question (34 million MIPS) [Lang94]. A practical real time CELP system must contain fast algorithms and take advantage of the data structures of CELP. The simple structure of a CELP analyzer is with fast algorithms changed into a scheme that is hardly understandable.

In the example implementation of CELP from DoD the spectral predictor is combined with the perceptual weighting filter and the code book searches are changed to a suboptimal combined search. The SCB shift 2 overlap and a preliminary pitch search (ACB) among integer delay only, are two other examples of reductions that can be used to make some of the calculations easier to do. The algorithm consists mostly of filter and energy calculations, correlations, convolutions, multiplications and additions. More information about CELP may be found in [Lang94] and in the DoD standard example program (DoDCELP) [DoDCELP].

4 Design process

To design this type of very complex systems in a quite short time, special efforts must be put into planning. We have been using project ideas from the course "Computer System Design Methodology" [CSDM] to run this project and the method of parallel step-wise refinement [Juhl] with parallel activities. Our time schedule with major milestones can be viewed in figure 6 and the results from the project including the time plan is further discussed in the conclusions, see section 8.



1. Specifications for RISC processor Dundee done
2. Width of datapath for coprocessor Mathilda fixed.
3. Clock-cycle true model of Mathilda working correctly
4. Complete C-model of Mathilda with CELP mapped on Mathilda
   Possible to listen to fixwidth calculated sound
5. Dundee complete
6. A/D D/A units done
7. Core part of CELP algorithm tested on the VHDL-model of Mathilda and Dundee
8. All done

Figure 6: Project time plan

4.1 Introduction

All functions from the specification have been implemented, and figure 7 shows how we have divided and assigned different tasks to hardware blocks.

The Control function is a real time system which runs the system thereby controlling the user interface, dialup, handshaking etc. The time constant for the control system is medium time and therefore not so critical. The CELP function is responsible for the speech coding of the sampled audio signal and is defined by the CELP algorithm which is very compute intensive. The data is sampled

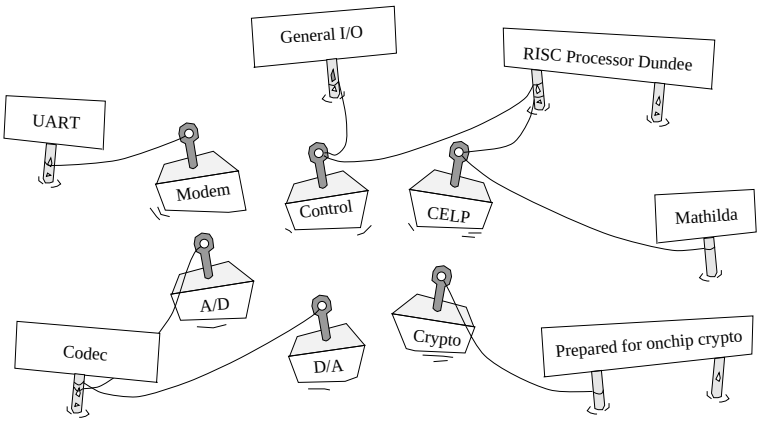


Figure 7: Functions tied to hardware

at 8 kHz and is compressed frame-wise making it a performance critical real time system.

The A/D and D/A converters are clocked at a fixed rate, sampling and restoring the audio signal and are not critical. The crypto function makes the data-stream between two units unreadable and is a black box in our design, so only bandwidth and interface are considered.

The whole ASIC is designed with the performance demanding CELP algorithm in mind.

4.2 CELP models

Early in the project a three step software model for the CELP algorithm was developed. The model should answer the following important questions:

- Is fixed point CELP possible ?
- How many bits are sufficient in the calculations ?
- How much memory is required for CELP ?
- How many cycles per CELP frame are necessary ?
- What primitives should be implemented in hardware ?
- And how about the speech quality ?

The software model is based on a CELP standard program written in C. This program was cleaned up and new functions and objects simulating our hardware were included.

The model showed that a 16 bit fixed point CELP is possible with a memory usage of less than 8 kword RAM and around 170.000 multiplications per frame to obtain sound quality. Results from this model were very important when Mathilda was designed.

4.2.1 Model I - DoDCELP

The DoD standard with the example implementation program was the main source of information. A second source of information was the program for the old product, SecuriVoice, which contains a Texas DSP. This program was written in assembler and used exactly the same calculation scheme as the standard source. With the help of those two programs, detailed studies of the CELP implementation could be done. It was also possible to make a preliminary calculation of how many mathematical operations of each type that had to be performed within each frame. Our plan was to convert all CELP calculations to vector operations suitable for a Multiply and Accumulate, MAC, datapath which can perform vectormultiply, -addition, -subtraction, correlations, energy calculations , convolution, filter calculations, block movement etc.

The software model showed that the most frequently used operations were correlation, energy calculations, convolution and vector multiply. These sources combined had enough information to verify that our idea of mapping the algorithm to a MAC unit was correct and could be done since we know what primitives must be performed and how fast.

4.2.2 Model II - MAC mapped CELP

Mapping of the CELP algorithm to a MAC unit was done by introducing a simple, not clock cycle true model implemented in a C++ class. This model was not pipelined and performed all necessary calculations immediately. Also a model of a RAM memory was added to make storage of variables as realistic as possible. We managed to map almost every math routine to work on a MAC datapath, only some small and not so heavy used routines were left unmapped.

This model showed us that a MAC unit could perform CELP, and from the model statistics the exact usage of every type of datapath operation, frame by frame could be extracted. This made it possible to calculate the cycle count and amount of memory needed in hardware. We could with this model optimize the mathematical operations performed by the datapath to suit CELP. We have inserted some tricks in the hardware to perform fast code book searching. See section 3.5 for more information about algorithmical improvements and section 7.4 for implementational details. At this point we removed the basic primitive vector addition from the datapath to save hardware because the operation was hardly used.

4.2.3 Model III - Fixed point CELP

To make the implementation of the datapath easier and more efficient the floating point calculations had to be converted into fixed point calculations. One goal was to achieve a solution where all calculations could be done in a 16 bit datapath, optionally with a wider accumulator. A new version of the MAC class that operates on 16 bit input, using a wide accumulator and producing 16 bit outputs was developed.

To make it possible to compare the fixed point compression results with the floating point compression results, all floating point calculations were kept in the program so that fixed and floating point calculations were done in parallel.

The RAM class was altered so that fixed point numbers and floating point numbers could be saved at the same address without overwriting each other. This model with parallel execution proved to be extremely useful during the hard and time consuming process of removing all mathematical overflows and underflows from the fixed point routines. By shifting the MAC results up and down depending on the expected numerical range of the results, nearly every underflow and overflow could be avoided. To avoid glitches in the sound if an overflow occurs, special hardware was added to the datapath to saturate the result in those cases. The algorithm is very sensitive since very small numerical differences will result in completly different output that sounds identical since the algorithm selects a vector depending on a best match scheme in which an almost infinite small difference may select a completly different code book index. The perceptual speech quality however is unaffected.

This model verified that a 16 bit datapath was producing compressed sound with adequate sound quality with approximately 170.000 multiplications per frame and 8k word of RAM. When this had been done the construction of the ASIC could be started.

4.3 Clock cycle true model

All digital parts of Darwin have been modeled using a clock cycle true model. In this model the design is partitioned into pipeline stages ie each clock cycle corresponds to a clock cycle in the final system. The model is also very useful for simulation because it behaves almost identically to the final system. The clock cycle true model can best be visualized using Werner diagrams, see [Werner].

Most tools do not support the clock cycle true model in a good way. There is however one, BBDS, which is totally based on VHDL, Werner diagrams and the paradigms associated with the clock cycle true model. BBDS is a graphical tool with which you can, and must, draw your design using Werner diagrams. This has several positive side effects one being the designer being forced to design in terms of pipeline stages and another making it possible for BBDS to check the design for common errors. BBDS also makes it possible to design using hierarchical pipelines. All components in BBDS are modeled using VHDL and BBDS also have a vast library of generic VHDL components, like flipflops,

multiplexers and gates, which are commonly used. The graphical editor of BBDS however is lagging behind the competitors.

BBDS has a very advanced coupling to industrial standard simulators and during simulation all values of all signals are viewed directly in the schematic. This greatly accelerates debugging the design. Another great feature of BBDS is the builtin synthesis cache which caches the area and timing of synthesized components. You are thus not forced to resynthesizes the entire design when changing a wire! BBDS is also capable of extracting and visualizing the critical path directly in the schematic.

4.4 Workflow

All blocks in the ASIC have been designed with a number of tools used together in the workflow drawn in figure 8

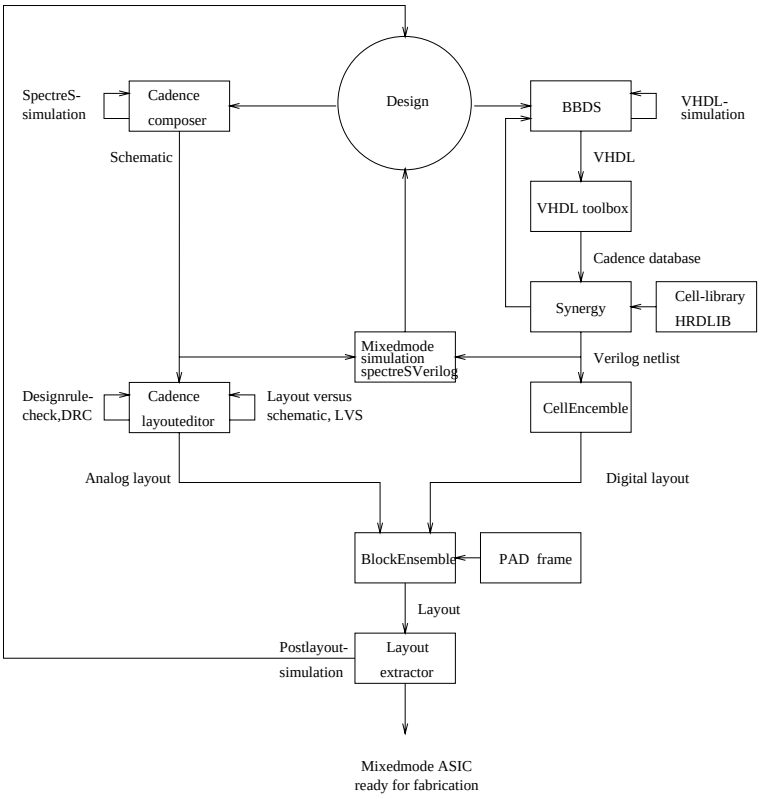


Figure 8: Workflow and tools used during design

First a schematic of all analog blocks was produced with the composer schematic editor. After that every analog block was simulated with spectreS, analog Spice simulation, to meet individual specifications. The analog layout was created in a layout editor and extracted, ie mapped to physical wires and components. It

is then checked with a Layout Versus Schematic, LVS, tool. This output was fed into an analog post layout simulator which simulates the design with models of physical devices including parasitics. Due to large simulation time only a small number of cycles, about 50, were post layout simulated.

The digital state machine was developed in BBDS [BBDS] which generated VHDL-files ready to include into VHDL-toolbox. After the files were imported, the netlister built a file structure later used by Synergy when synthesizing to the 0.8μm cell library HRDLIB. The output from the synthesizer is a Verilog netlist which contains interconnected instances from HRDLIB. This Verilog file is then included into Cadence as a schematic and simulated with the mixed mode simulator spectreSVerilog.

A layout containing analog and digital parts may be post layout simulated with an analog transistor based simulator, but to cut simulation time for large designs it is better to use a mixed mode simulator which have not been possible due to problems with Cadence.

5 Implementation

Our implementation is based on the earlier product SecuriVoice and on the DoDCELP with our own mapping to another type of calculation unit. The SecuriVoice CELP is executed on a TMS320C3x floating point DSP at 33MHz and the major part of the CPU time is used for the speech compression. The only algorithmical difference between SecuriVoice and DoDCELP is that the SecuriVoice uses inverse convolution in the SCB-search. Some other standard optimizations widely used in assembler programs are also present.

In our implementation CELP analyzer/synthesizer, Crypto/Decrypto, AD/DA, system control and user interface control must fit in one custom ASIC with external ROM and RAM. Early in the project a solution with a RISC processor called Dundee and a coprocessor called Mathilda was chosen. Mathilda is a dedicated vector processor doing multiplication and accumulation, MAC, operations. If some general input/output ports were added, the RISC processor could be used both for algorithm control and system control, removing the demand for a special control unit. Dundee's instruction set was designed to be as minimal as possible still supporting all necessary instructions to make an effective C/C++ compiler. Both codesize of the application running on Dundee and the core must be small, thus 16 bits instructions and 16 bits registers were used.

Dundee and Mathilda share the same data memory together with an on chip crypto unit, Croco. This is implemented with a small bus arbiter solving bus conflicts. To archive high performance Dundee fetches its instructions from a private ROM keeping the pipeline filled even if Mathilda is running a memory intensive calculation.

To make the hardware manageable a fixed point implementation of Mathilda was preferred and the fixed point model of CELP showed that this was possible. The block schematic of the complete chip is shown in figure 9.

According to the US Federal CELP standard 1016 [Std1016] a 12 bit A/D- and D/A-conversion should be used with a 8 kHz +/- 0.1% samplingrate. To make the design cheaper 10 bit converters was used. The CELP algorithm introduces some noise and distortion into the speech signal, so a 12 bit converter is not really motivated. To this adds the fact that a simple fast 12 bit on chip converter with an error of maximum 1/2 LSB is very hard to achieve without laser trimmed components.

In this project the CELP algorithm is only used to minimize the transferred information between identical crypto boxes. Therefore the CELP standard needs not be followed precisely and the decision to use only 10 bit converters can be accepted.

To open up the possibility to remove the external modem circuit and replace it with software running on Dundee, the converters can operate at a sample frequency up to 16 kHz and the number of channels has been doubled. These extras add a minimum of components to the analog block because of the possibility to time share function blocks. If those extra channels are unused they

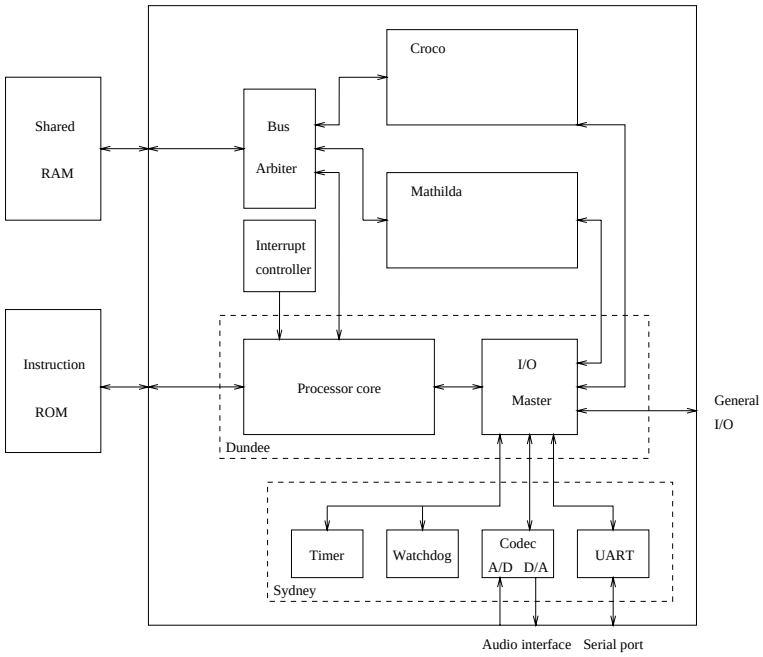


Figure 9: Block schematic of chip

can be left unbonded in the final package and two pins are saved.

5.1 Pin budget

One of the most important design issues for a digital ASIC is how to use the available pins on the package. This number is a hard limit and it is generally not possible to order a package with more pins because of the added cost. Time sharing can be used but this may be restricted by the operating frequency or power consumption.

In our design an approximative pin count looks like table 1. From this table a preliminary power consumption can be calculated if the specifications for the pads are known. We do not have this information but a rough estimate is:

A Pad capacitance of 20 pf and a bus fan out of 4.

$(Fanout + 1) * Pincnt * C * V * f = P_{disipation}$

High-speed signals:

$(4 + 1) * 45 * 20pF * 3.3V * 28MHz \approx 0.5W$

Low-speed signals:

$(4 + 1) * 32 * 20pF * 3.3V * 9MHz \approx 0.1W$

This indicates that it should be possible to fit this chip in the selected package.

Port/Usage	Pin count	Frequency
Shared RAM interface	41	28 MHz
Dundee ROM	32	9 MHz
Gen I/O Keyboard	9	$<$ 1 Hz
Gen I/O LCD	11	100 kHz (Burst)
Gen I/O SmartCard	4	100 kHz (Burst)
External Interrupt	1	10 kHz
Bus interface	4	28 MHz
Reset	1	-
UART	8	10 kHz
A/D,A/D	9	16 kHz
	120	
Vdd/Gnd	19	-
Clock	1	28 MHz
Total	144	

Table 1: Early estimate of pin usage

6 Implementation of the analog block

The block that was the hardest to design was the DAC which was used by all channels. The block should operate at medium speed with almost rail to rail specifications. First a 10 bit binary switched resistor stage was evaluated but the area of the layout was too large. Then a two stage 4 + 6 bit design was evaluated and proved to meet our requirements perfectly.

Finally a test chip with the AD/DA module was prepared and sent to fabrication, see figure 10.

Figure 10: Plot of AD/DA test chip

6.1 Design overview

In figure 11 an overview of the analog design is drawn. The block marked Main Control is the digital state machine responsible for the cycle control of all different channels and the communication with Dundee. The Successive Approximation Control contains the logic that handles the successive approximation iteration used by the input channels. The D/A converter is the largest analog sub block which converts a 10bit digital word to an analog voltage. The conversion time specification ($62.5\mu s$) for the AD/DA block was slow enough to use an area saving design, where all four channels were time sharing the same D/A converter.

One 10 bit A/D conversion occupies the D/A converter block during 10 cycles and a 10 bit D/A conversion uses the block one cycle. Two A/D channels (A,B) and two D/A channels (C,D) only occupy 22 cycles out of 32, those extra could then be used to relax the specification for the internal S/H system that remembers the analog voltage for output channels C and D. Maximum sampling rate is limited by analog components in the circuit and all analog blocks are designed to run up to 16kHz with a 0.5MHz bit clock.

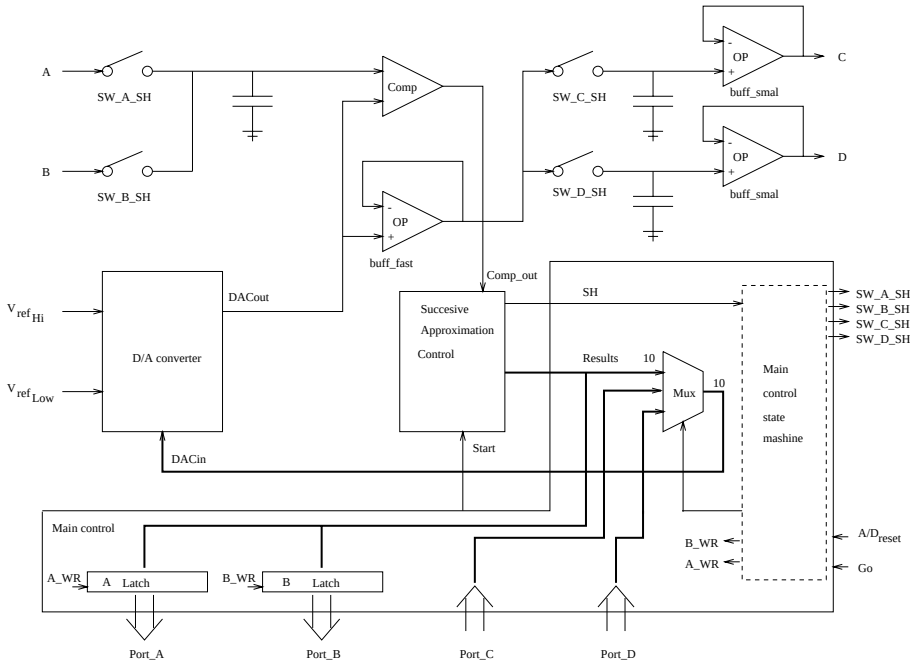


Figure 11: Overview of analog block - the complete A/D, D/A unit

6.2 A/D Successive Approximation Control

The successive approximation, SA, iteration starts when the signal Start goes high by loading the memory cell with the first guess "1000000000". In the next

cycle a higher/lower signal, is available and a decision about the next cycle is taken. If lower the new guess will be "0100000000" and if higher the guess will be "1100000000". After 10 cycles the conversion is done and the result is sent to Dundee.

6.3 Analog design

Op-amps are based on the standard dual stage circuit shown in figure 12. A differential input stage with single output fed to a source follower are used. For detailed information about op-amp design see [Mol94] and [MOSAmp]. This simple circuit is easy to realize but both input and output stages have problem with rail to rail signals and in this design all amplifiers are used very close to the limit.

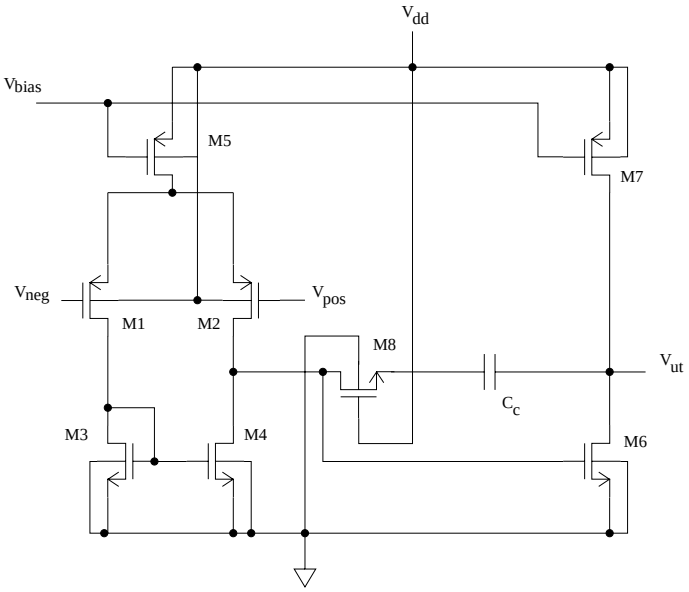


Figure 12: Operational amplifier standard schematic

6.3.1 Buffer amplifiers

At certain points in the design signal stages must be isolated from each other with buffers. In the schematic two types of buffers have been used as described in figure 11. Buff_small is used when the load of the amplifier is small and buff_fast is used when a large sample and hold should be charged quickly. Design calculations and final transistor ratios can be found in appendix C.

Specifications for buff_small:

Phase margin is specified with M, which is the distance between G_B and pole number 2. $M = 3 \Rightarrow \phi_m = 90 - arctan\frac{1}{M} \approx 72°$

Gain bandwidth product, $G_B = 2\pi \cdot 0.25 \cdot 10^6 rad/s$

Slewrate, $S_R = 0.5V/\mu s$

Load capacitance, $C_L = 1.5pF$

Supply, $V_{DD} = 3.3V$

Specifications for buff_fast:

$M = 3$

$G_B = 2\pi \cdot 0.5 \cdot 10^6 rad/s$

$S_R = 2.75V/\mu s$

$C_L = 2.8pF$

$V_{DD} = 3.3V$

6.3.2 Comparator

The comparator is an uncompensated op-amp with one extra inverter stage to speed up the transition. To make the propagation time low, small transistors are used and the input transistor bulk is floating.

In use, the comparator is loaded by the digital SA-block containing 10 small transistors $< 0.1pF$. The propagation time must be less than

$$\frac{1}{16ksamples/s \cdot 32cycles} \Rightarrow 1.95\mu sperbit$$

The slew rate must with $V_{dd} = 3.3V$ be ten times faster to get a fast flank.

$0V \rightarrow 3.3V$ in $1.95\mu s \Rightarrow 1.68V/\mu s \Rightarrow S_R = 16.8V/\mu s$

The input offset error must be $< 1/2$ LSB and the circuit should be fast because the higher/lower decision must be taken in a short period of time. The comparator need no compensation because it is used in analog open loop configuration. The output swing should be 0.75 V from each rail. The gain must be at least $2^{10} = 1024$ times. Calculations and transistor ratios can be found in appendix C.

6.3.3 Sample and hold

The schematic of a simple sample and hold circuit can be viewed in figure 13. When the transmission gate is activated the circuit follows the input signal by charging and uncharging the sample and hold capacitance. The signal is held when the transmission gate is deactivated. In figure 14 the curve form of a sample and hold event with common errors are illustrated. The voltage

drop of the S/H capacitance is incredibly small, because of very high input resistance of CMOS transistors ($R_{in} \sim 10^{18}\Omega$) and is therefore no problem. A high-resistance transmission gate in combination with a large S/H capacitance decreases the ability to follow a fast changing input signal. In our case the input signal range is from 0.5 V to 2.5 V so both n- and p- type transistors are used. The series resistance for an activated transmission gate with minimal dimensions is about 10 kΩ and shows a small voltage dependence.

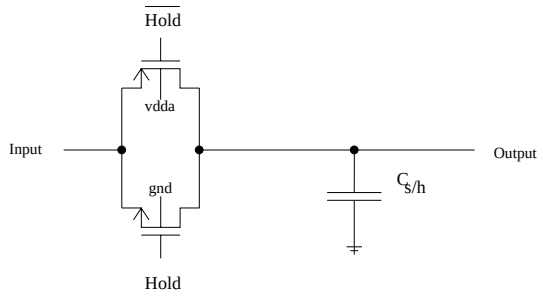


Figure 13: Schematic of basic sample and hold circuit

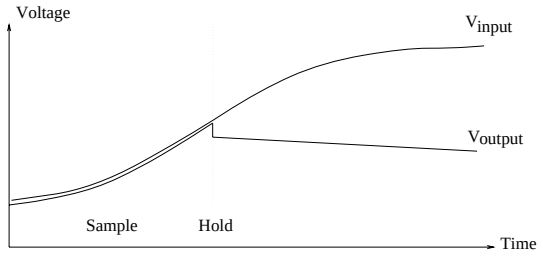


Figure 14: Example of sample and hold curve form

A large problem with the sample and hold circuit is the error introduced at the point where the circuit starts to hold the signal. All MOS transistors have gate-source and gate-drain parasitic capacitances. If the gate signal changes rapidly and for example the source is connected to a node with non finite capacitance to ground, the voltage level of the source node will change. This problem can be solved by increasing the capacitance at the source node, selecting a smaller transistor with less gate-source capacitance or charging a new node with same parasitic coupling to the source equal but in the opposite direction. In our case the fast changing gate during the hold event is responsible for the small but important voltage level error. If minimal size of all transistors and a large sample and hold capacitance are chosen the error will still be within 1/2 LSB and therefore a slightly different circuit is used, figure 15. This circuit is inspired by [Yuan95] but with only one side of the dummy transmission gate connected to the S/H node.

In this circuit a dummy transmission gate is used to compensate for parasitics in the real transmission gate. The parasitic capacitance from the hold signal to the output node is almost the same as from the inverted hold signal to the

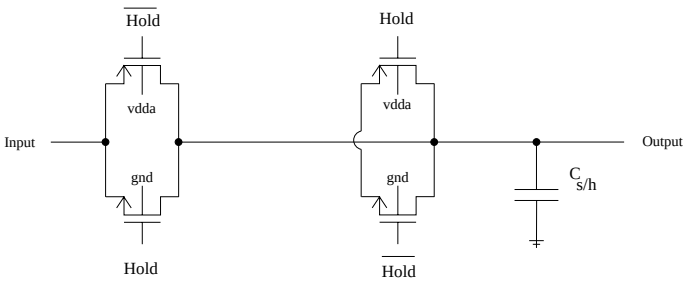


Figure 15: Improved sample and hold circuit

output node. If both control signals change at exactly the same time almost no change in the output node voltage can be measured. In reality a small time difference between the two control signals results in charge injection and a small voltage error.

To minimize the layout the S/H capacitance is implemented as a gate capacitance, see figure 16. The capacitance shows a large voltage dependency with up to a factor 5 drop relative to C_{max}. This large change may be smoothed a little if both types of transistors are used. The selected transistor size is $50 \cdot 50\mu^2$. A DC operation point calculation with Cadence shows a C_{max} of 2.7 pF for each transistor.

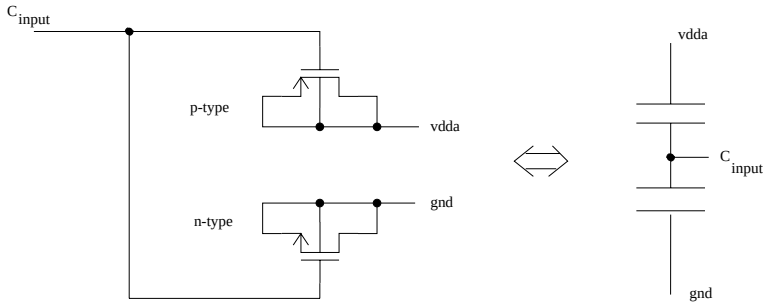


Figure 16: Sample and hold capacitance

6.3.4 D/A converters

Digital to analog converters may be implemented with many different techniques ranging from simple resistor stages to multi-phase charge-coupled capacitance networks. Depending on the principle of operation, D/A converters can be divided into the following classes

1. Parallel D/A converters
2. Weighted resistor/capacitors/current D/A converters

3. Algorithmic D/A converters, ie $\Delta - \Sigma$

4. Pipelined D/A converters

A parallel converter uses a voltage divider resistor string and a network of switches to divide a reference voltage into 2^N different steps, figure 17. The chip area is large for N $>$ 8 and the settling time may be long due to large parasitic capacitances. The stage must be buffered if loaded and this type is only practical up to a size of 8 bits. One advantage is that it provides monotonicity of the transfer characteristic. Another way is to use a tree of switches to decode to output signal like the one in figure resistordac2.

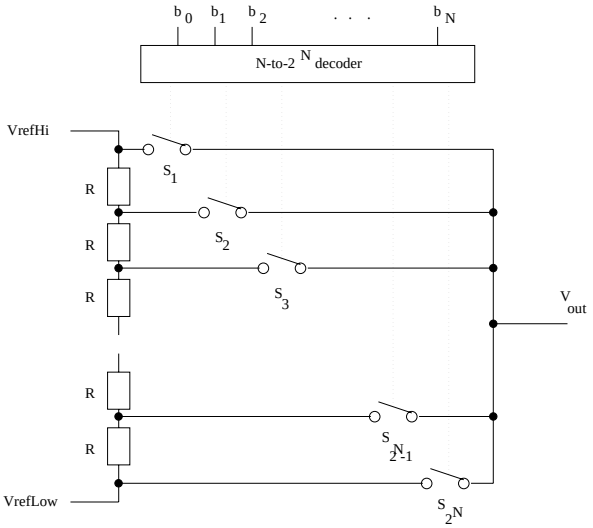


Figure 17: $2 - to - 2^N$ Decoded resistor D/A converter

Weighted D/A converters can be constructed with several types of elements used to binary divide and later add together the final results. Current is divided with current mirrors, voltage with resistors and charge is divided with a capacitor network. An example of a $R - to - R^N$ resistor stage can be found in figure 19 and in figure 20 a $R - 2R$ resistor stage is shown.

The algorithmic D/A converter, also called a serial or cyclic D/A converter operates with bit streams. N clock cycles are required to convert one N bit word of data. A pipelined D/A converter consists of N equal stages connected after each other to perform one bit of conversion per stage resulting in a structure capable of converting 1 word of data in 1 cycle, after the pipeline is filled that is.

Algorithmic and pipelined converters are too complicated for this application but are useful when high speed and/or high resolution are required. In this application a simple 10 bit medium speed D/A converter is needed and therefore a resistor based D/A converter with straight forward function and no special clocking requirements is selected.

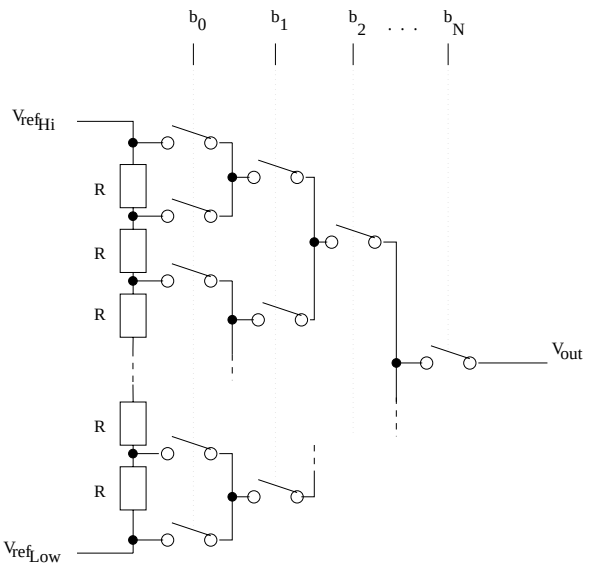


Figure 18: Tree decoded resistor D/A converter

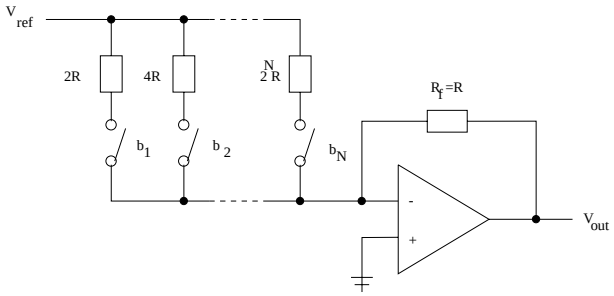


Figure 19: Binary weighted resistor D/A converter

If two different voltage references are used, V_{out} from the D/A converter will be

$$V_{out} = \frac{V_{ref_{hi}} - V_{ref_{low}}}{2^N}(n - 0.5) + V_{ref_{low}}$$

$$LSB = \frac{V_{ref_{hi}} - V_{ref_{low}}}{2^N}, N = 10$$

$$V_{out} = \frac{2.5 - 0.5}{1024}(n - 0.5) + 0.5V$$

$$LSB = 0.001953V = 1.953mV$$

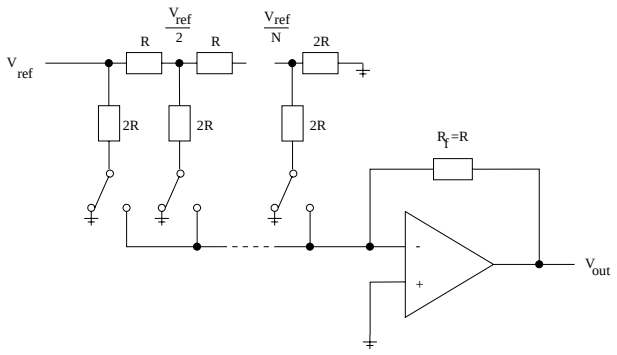


Figure 20: $R-2R$ resistor D/A converter

6.3.5 Resistor stage D/A converter

Due to the low supply voltage (3.3 V), lack of negative voltage, large voltage swing (2.0 V), medium speed conversion time (2 μS) the selection and development of the D/A converter was rather complicated. Some basic circuits were immediately eliminated.

The binary weighted resistor D/A converter viewed in figure 18 would have been too large $> 4mm^2$.

A R-2R resistor D/A converter like the one in figure 19 is not possible to use since the output signal from the operational amplifier is negative.

In the selection process a non negative R-2R stage was developed. If the positive input of the op-amp is connected to a new reference voltage V_{mean} the virtual ground at the input is raised to V_{mean} resulting in a non-negative output. A closer look at the schematic in figure 21 shows switches with internal resistance in series with resistors in the stage making it impossible to do the layout of all resistors with reasonable size. The switch size can be increased to decrease the resistance, but only up to a point when the problem with increasing capacitance slows down the stage to much. Dummy switches in the circuit have been tested but without success.

Binary weighted resistor D/A converters as in figure 19 are hard to design since the resistor values must be much greater than the internal series resistance of the switches. This becomes even more difficult as N become large due to the area of the huge resistors.

To remove the problem with switches, operation amplifiers can be used to create a stage with active switchable references. Unfortunately this design is sensitive to the numerical value of R, since many op-amps are connected to almost the same point. The layout area is large and the output depends on op-amp V_{offset} which may be tunable if all op-amps have the same V_{offset}.

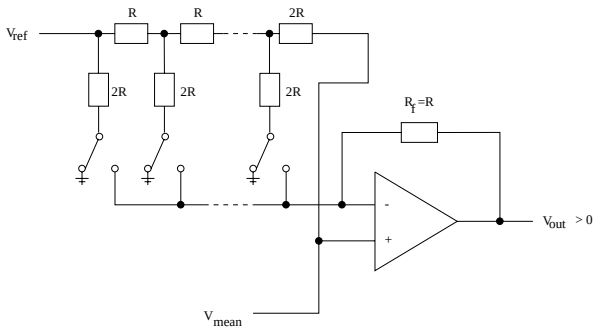


Figure 21: Non negative $R-2R$ resistor D/A converter

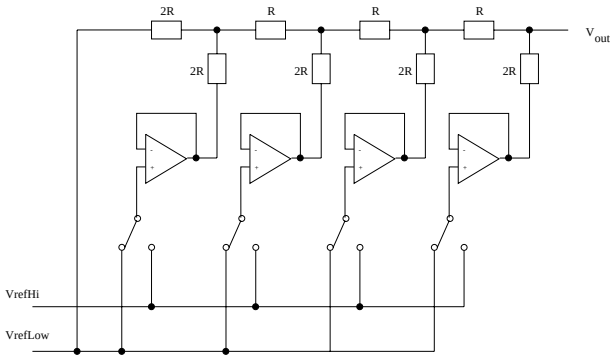


Figure 22: Active reference D/A converter

6.3.6 Multiplying resistor stage D/A converter

If two stages with A and B bits are connected in cascade to each other the total stage will have A+B=N bits. Stage one selects reference voltages for the second stage which splits the references into even smaller parts. This configuration is called a multiplying D/A converter. Almost all types of converters can be used if they can be connected together without interference because A and B are less than 8. Optionally a buffer amplifier can separate the two stages from each other.

6.3.7 Selected stage configuration

The best results where obtained with a $4+6$ configuration including a 4 bit resistive string converter to feed $V_{ref_{hi}}$ and $V_{ref_{low}}$ to a 6 bit resistor string converter, see figure 23. Since the first stage must select two different voltages located next to each other, every selectable node in the stage has got its own switch connected to an odd/even network. A crossbar switch ensures that the high reference always has a higher potential than the low reference. All switches are controlled by a logic block which turn them on pairwise. Two buffers are

inserted to isolate the first stage from the second stage. The capacitive load of those buffers is quite large, $1.5pF$ each and therefore two buff_fast are used.

Voltage over stage one is $V_{ref_{Hi}} - V_{ref_{Low}} = 2.0V$. A current of $10\mu A$ is chosen, resulting in a total resistance of $200k\Omega$ divided into sixteen $12.5k\Omega$ resistors. Over stage two the maximum voltage is $0.125V$, ie the difference between two neighbour nodes. The current is selected to $5\mu A$ and flowing through 63 resistors of approximately 390 Ω each.

A simulation of the converter shows some difference from the ideal value traceable to offset errors in buffer amplifiers.

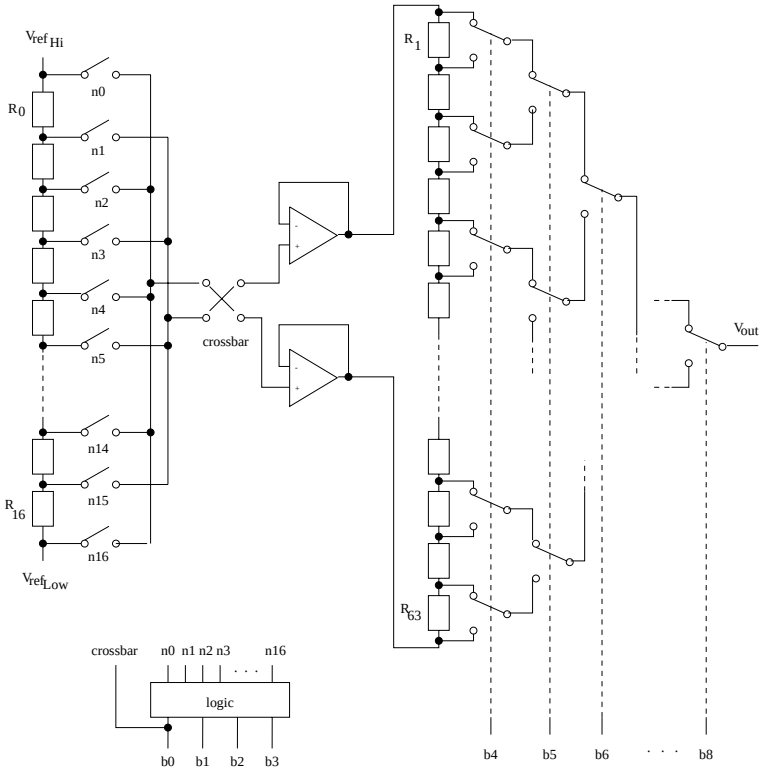


Figure 23: Schematic of the 4-6 A/D converter

6.3.8 Bias-generator, bootstrap type

In an analog circuit some MOS transistors need a constant input reference voltage in order to obtain the right working point. This voltage is measured relatively to the V_{dda} rail and should be as stable as possible, ie independent of supply voltage or load. This reference voltage may be created off chip but normally and in our case this voltage is created with a special bias generator block, see figure 48 in appendix C . This technique is called a V_T-reference source or a bootstrap reference.

The V_{out} output is only connected to transistor gates controlling current generators in different transistor stages. All gates have high capacitance but almost no leak current. Since the load of the circuit is very small, it should be possibly to generate a very stable voltage.

6.3.9 Vref-generator

The D/A converter in the circuit needs two different reference voltages to operate properly, $V_{ref_{low}} = 0.500$ V and $V_{ref_{hi}} = 2.500$ V. These two reference voltages are critical if high absolute resolution from the D/A converter is required. $V_{ref_{low}}$ and $V_{ref_{hi}}$ must then be generated off chip with very stable reference circuits. In cheap applications it may not be necessary to use stable references and therefore two on chip references with are included in the design.

These references are generated with the help from the Vbias circuit. The current in the output stage of the Vbias generator is mirrored to current generators and resistors for current to voltage conversion. Both $V_{ref_{hi}}$ and $V_{ref_{low}}$ is measured relative to ground and should be stable.

The load on $V_{ref_{hi}}$ and $V_{ref_{low}}$ is large ($10\mu A$) because they are connected directly to the resistor stage. They handle this large load by using higher current, see figure 24.

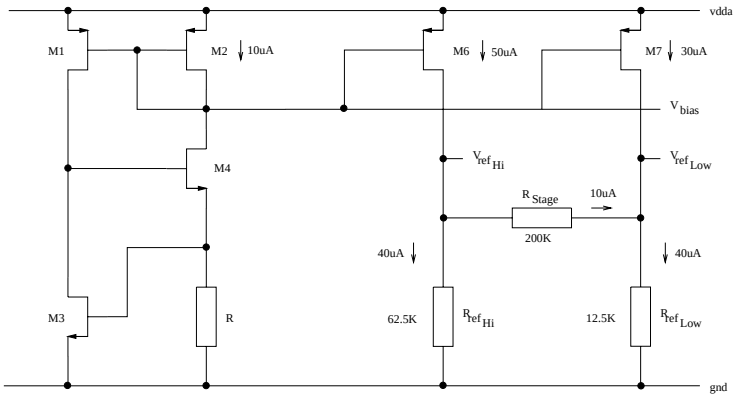


Figure 24: Schematic of Vref circuit

$R_{ref_{hi}}$ and $R_{ref_{low}}$ resistors are not implemented on chip. This opens up the possibility to fine tune them in production.

The Vbias signal follows the V_{dda} rail nicely when V_{dda} changes but the reference circuit is not as good because $V_{ref_{hi}}$ is so close to V_{dda} and $V_{ref_{low}}$ is so close to ground.

Small current changes in the current mirror are responsible for a part of the error. Non ideal current generators (M6/M7) are responsible for the relatively large error in the forwarded current copied from the current mirror.

6.4 Simulation of A/D converter

All blocks in the A/D converter have been simulated and checked against their specifications. Digital blocks containing logical functions have been simulated with a VHDL-simulator and after synthesis with a mixed mode simulator and finally a few clock cycles have been tested with an analog post layout simulator.

Analog blocks are harder to simulate and verify because they have a more complex behavior. All analog blocks have been simulated in testbenches with surroundings as close as possible to the real environment. Mixed mode simulations of the complete A/D converter block have been done. Since the simulation time was long only simulation of single cycles was realistic. According to those simulations the A/D converter should work, but it is uncertain if all specifications are met. The design may have some problem with accuracy and rail to rail operation. No block has a larger simulated error than 1/2 LSB.

It has not been possible to run all simulations and calculations we wanted to do because of the bad post simulation plot and calculation environment. Due to lack of time it has not been possible to filter out the interesting data from Cadence for a more advanced post simulation analysis.

6.5 Prototype production

The CODEC has been sent to fabrication. To save some area output signals are multiplexed and some input signals have been connected to more than one input. The floorplan of the test chip can be viewed in figure 25 and a plot in figure 10. The test chip had not arrived from fabrication when this report was written.

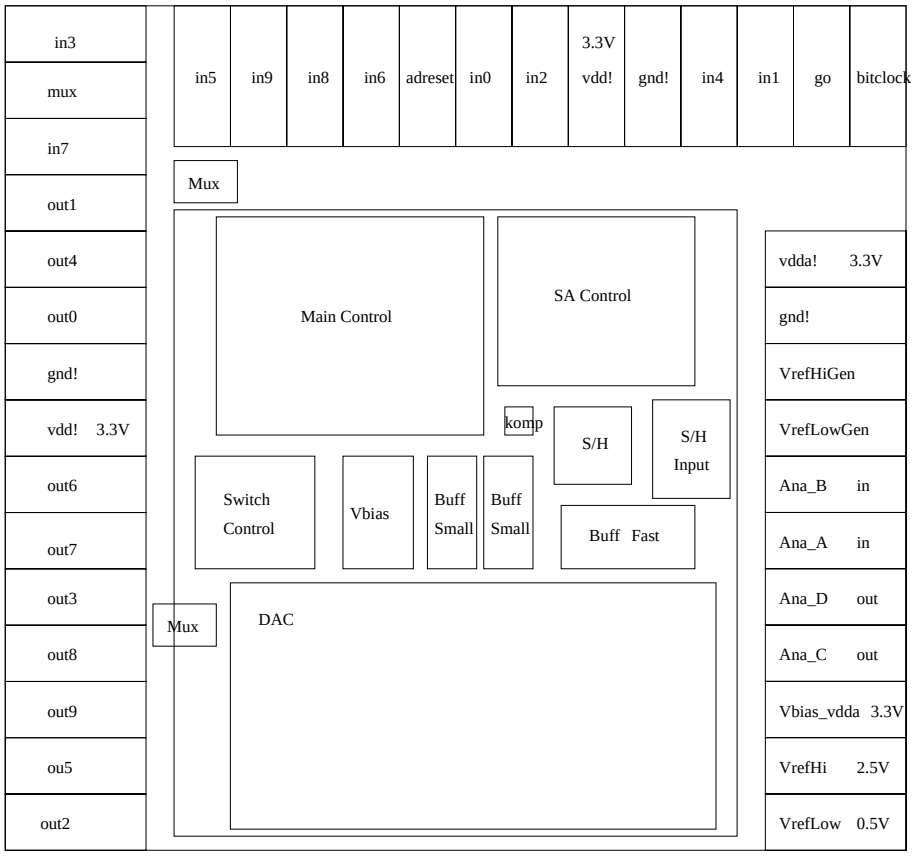


Figure 25: Floorplan of AD/DA test chip

7 Implementation of digital block

There are several constraints associated with digital design of which the most important are area, speed and pad count. In this design the area was limited to about 30 mm^2. The speed, ie clock frequency, was bound by the CELP algorithm and the available memories. To be able to process the CELP algorithm in real time the chip had to perform 6 millions multiply-and-accumulate, MAC, operations per second. The memories had a cycle time of 45-60 ns for ROM's and around 12 ns for RAM's. The pad count of the final chip couldn't be higher than about 144 pads. Since the chip would have a lot of IO-ports and due to the large difference in cycle time between the different memories, an architecture with two 16-bit memory ports, one for ROM and one for RAM, was chosen, see section 5. The large difference in cycle time between the memories also led to the decision to let the RAM cycle time be exactly one third of the ROM cycle time.

All digital parts have been written in pure VHDL using a graphical tool called BBDS. The VHDL has been synthesized by Synergy and placed and routed using ordinary Cadence tools, ie composer and cell ensemble.

7.1 Bus arbiter

Because a solution with only one RAM was chosen and many units needed to access this single memory, a bus arbitration scheme must be used. The bus arbiter is responsible for enforcing this scheme which is rather simple and uses a static priority order where each unit has a unique priority. The attached units and their priorities are:

- External bus master (highest priority)
- On-chip future crypto unit
- Dundee microprocessor
- Mathilda DSP-unit (lowest priority)

The external unit can be another Darwin, a DMA channel or controller, or another CPU. This makes it possible to couple several Darwins together if necessary.

All units use an interface similar to the bus arbitration scheme used by the 680x0 microprocessor, see [m68k], series made by Motorola. It uses two handshake signals named Bus Grant, BG, and Bus Request, BR. The attached units must provide an address bus, a write signal and an optional write data bus, which is only needed when the unit wishes to write to memory. The memory cycle is pipelined into three stages, the arbitration stage, the access stage and the data stage, see figure 26. The unit that wishes to use the memory asserts BR in the arbitration stage, during which the address, write signal and data bus must be valid. If the unit acquires the bus, the bus arbiter will assert the corresponding

BG signal at the end of the arbitration stage. In the access stage the actual memory access is made and in the data stage, which is only present when the access made was a read access, the read data is available to the accessing unit.

When a unit has acquired the bus, it owns the bus until it releases its BR signal. There is however an exception to this rule: Mathilda, the DSP-unit, never owns the bus longer than 1 clock cycle.

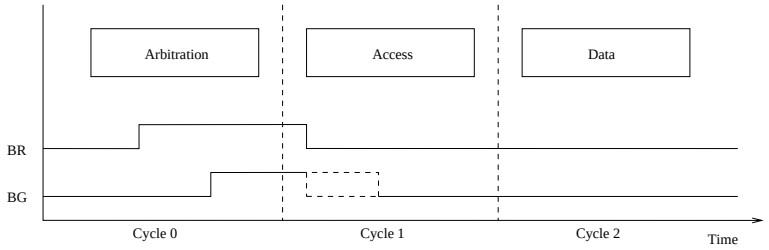


Figure 26: Timing diagram of the bus arbiter

The Bus arbiter is itself implemented using a FSM and simple multiplexers, see figure 27.

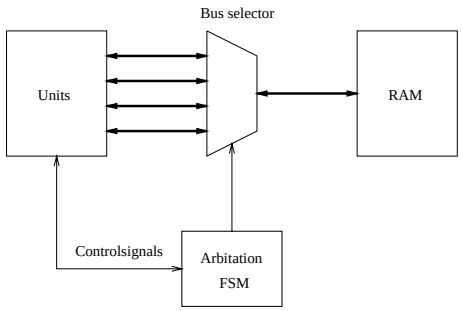


Figure 27: Sketch of bus arbiter

7.2 Dundee

Darwin needs a small controller that runs the show. Although the present control needs of the SecuriVoice is rather limited this is likely to change much depending on the fact that Mathilda cannot run the CELP algorithm by itself but must be controlled by a microprocessor. A small yet efficient microprocessor must thus be designed. The obvious design solution was a RISC design with a load-store architecture because of its simplicity and high performance. Furthermore the design had to be small and code efficient. Thus a solution with 16-bit instructions and 16-bit register width was chosen. The instruction set was influenced by the M68000 microprocessor and the design tradeoffs discussed in [ComA]. There are some complications associated with using a 16-bit RISC, memory access can be ineffective and constants may become a problem.

To solve these problems to some extent, two registers are combined to form a 20 bit pointer and also variable length instructions are used for immediate constants.

7.2.1 Programming model

Like most RISC's the programming model is quite simple, 16 general 16 bit registers and up to 16 special register. The special registers are used for multiply and divide instructions, interrupts and system control. The memory model is however a bit more complex. There are several memory areas divided into two classes, normal memory and alternative memory. The normal memory class is the ordinary data memory while the alternative memory class consists of external memory access, both atomic and non-atomic, memory mapped IO access. Different memory instructions are used for the two classes. The alternative memories cannot be larger than 64 kword, or 128 kbytes, while the normal memory can be as large as 1 Mword or 2 Mbytes. Up to 1 Mword ROM, ie instruction memory, can also be used.

7.2.2 Instruction set

The instruction set is designed to be able to support C/C++ with a minimum number of different instructions, see appendix A. The only memory addressing mode supported is register indexed with a constant. Condition-code flags are used, although they give higher clock cycle time they also give a more simple programming model. Supported instructions are:

- Normal addition and subtraction which use three registers
- Constant add instructions
- Basic logic functions
- Load and store instructions
- Instructions that use the special registers. Such as: move to and from special registers, multiplication and division.

7.2.3 Interrupts

Only one interrupt handler is supported in the Dundee architecture. Multiple interrupt sources are supported, although without priorities. The individual interrupt sources can be masked and pending interrupts can be read from a special register. The interrupt handler can thus easily decide what source triggered the interrupt. One external, ie off chip, interrupt source is supported. This can be used to couple several Darwins together.

7.2.4 Pipeline

Dundee is implemented with a four stage pipeline.

- IF: Where instructions are fetched from ROM.
- ID: Here the instructions are decoded. Operands are fetched from the register bank, constants are evaluated and branches are executed.
- EX: Finally the instructions are executed. Memory is accessed.
- WB: The result is written back to the register bank.

For a more complete description of the pipeline see figure 28. The memory controller is responsible for accessing the correct type of memory depending upon the used memory area.

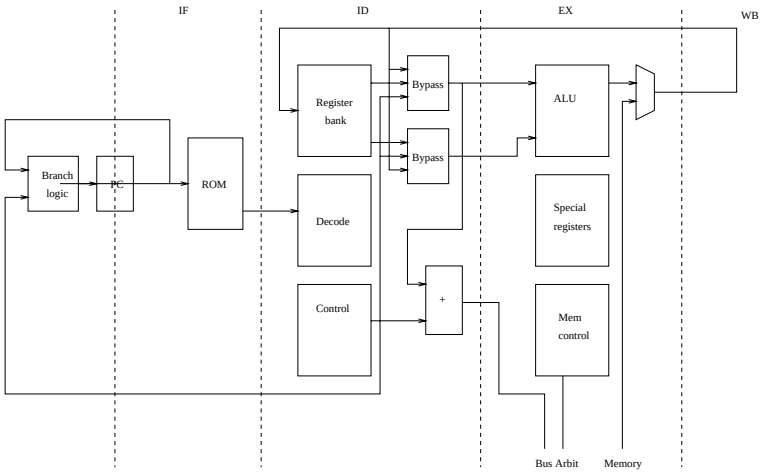


Figure 28: Block diagram of Dundee pipeline.

Because Dundee operates at one third of the global clock it can make a complete memory access during the EX stage and no additional memory stage is needed.

Dundee has full bypass thus no stalls are inserted when a data dependency is encountered. It also has delayed branch with one branch delay slot, ie the instruction after the branch is always executed whether the branch is taken or not. The instruction placed in the branch delay slot must however not use any long constants or be a control instruction, ie be a branch, jump or any other instruction which might alter the program counter. The pipeline stalls one cycle when a long constant is used in an instruction because one additional ROM access is needed to fully fetch the instruction. Dundee also stalls if it accesses memory but does not become master. In all the CPI of the pipeline is expected to be around 1.3 average. No real measurement has been done because no instruction statistics are yet available.

Interrupts are only handled imprecisely when the processor is in a safe state, ie is not stalling or executing a branch delay slot. This greatly simplifies the interrupt handling since the state of the pipeline does not need to be saved. This works in all cases because no exception can occur when executing the instructions.

7.3 Sydney

Syndey is a loose collection of IO-ports, UART, interface to the CODEC and timers. Syndey supports up to 28 bits of bidirectional IO where the direction can be individually controlled for each bit. It also has two dedicated 10 bit IO-ports that interface to the CODEC. It also has one fixed watchdog timer and three variable timers, see figure 29.

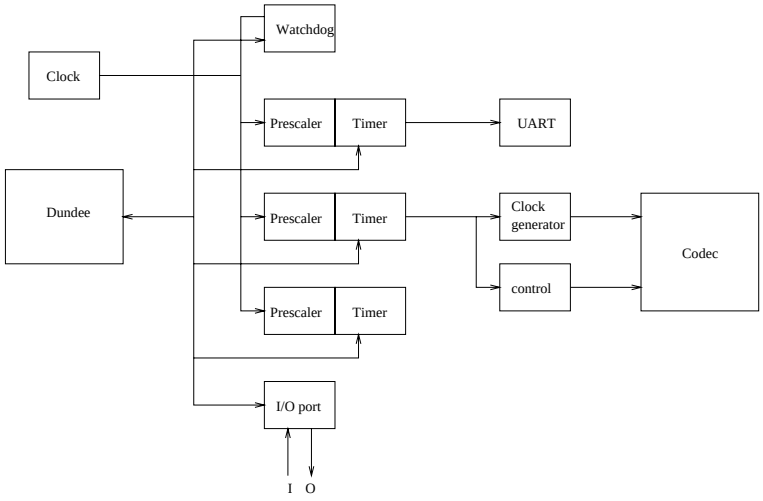


Figure 29: Block diagram of Syndey

The bidirectional I/O ports are used to connect a small (2x20 character) LCD display, telephone style keyboard with 16 keys and a SmartCard. The LCD needs 11 lines, the keyboard 9, the SmartCard 4 and the serial interface uses 3 I/O pins.

Always remember, no chip has enough general I/O!

7.3.1 Timers

The watchdog timer, if enabled, generates a reset if it has not been polled during the last half second. At reset the watchdog timer is disabled and it is possible to disable or enable it at any time.

The three variable timers can all give interrupts to Dundee. One of them is meant to be used as a general low frequency timer. It can be used for time

handling and context switching. The other two are specialized for controlling, ie generating the clock signals for the UART and CODEC.

The variable timers all work the same way. The global clock is prescaled by a constant which is different for each timer. The prescaled clock is then used to clock a variable timer which is made of an accumulator, see figure 30. The generated carry is used to trigger the interrupt control logic.

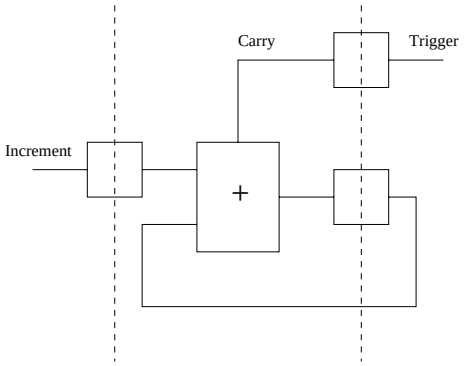


Figure 30: Block diagram for the variable timer

7.3.2 UART

The UART transmits and receives asynchronous serial data over a standard RS-232 serial interface. The baud rate, or the communication clock, is controlled by one of the variable timers. This makes it possible to communicate at a number of different rates.

This UART has a one byte buffer in both directions and the input interface is connected to an interrupt line. A block diagram showing both receiver and transmitter is present in figure 31. The status register is readable from the CPU to check UART status. To this register 3 extra signals have been added (bit 2, 3, 4) that may be used as a general input port if the UART is unused.

Bit	Name	Description
0	Empty	Transmit register empty
1	Full	Receive data ready to read
2	DSR	Input connected from DSR signal (Copy)
3	CTS	Input connected from CTS signal (Copy)
4	RxD	Input connected from RxD signal (Copy)

Table 2: UART status register

One byte is received by first waiting for a start bit, waiting one half bit more and checking the start bit again, waiting one bit delay, reading/waiting for 8 data bits and finally checking the stop bit. The stop bit is checked and if missing the data is discarded.

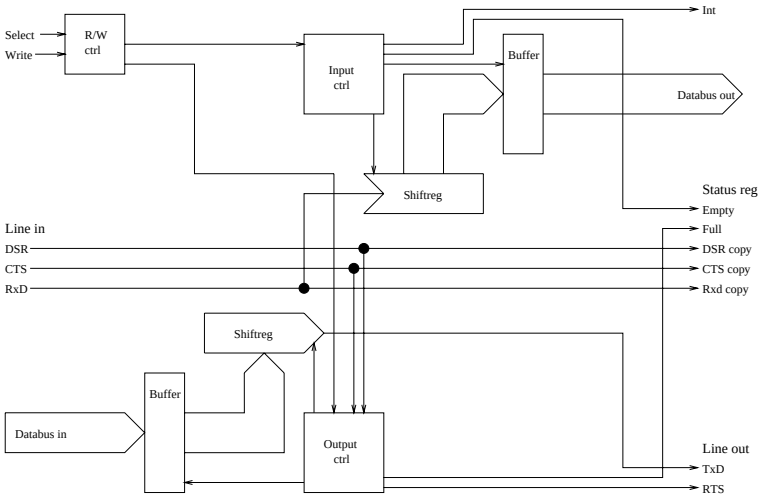


Figure 31: Overview of receiver and transmitter

7.4 Mathilda

Mathilda is by far the most complex part of Darwin. It uses up nearly half of the total chip area and has taken most of the development time. Mathilda is a stand-alone DSP unit which works very much the same way as the hardware implementations of the BitBLT algorithm, commonly called "blitters" which are used in video hardware. Mathilda can on its own perform common DSP primitives like filters, correlations and energy calculations. It is based on the fact that most DSP calculations is basically a variation of equation 1.

$$d_j = 2^k * \sum_{i=0}^{m-1} s_0(i,j) * s_1(i,j), j = 0..n-1 \quad (1)$$

Where k,m and n are arbitrary constants.

Mathilda is essentially a datapath optimized for multiplication followed by an accumulation. However since the most compute intensive operations is the stochastic code book search, see section 7.4.4, an additional datapath is also supplied which uses the nature of the values in the stochastic code book and the structure of the code book to make the search parallel. Usually Mathilda saves signed 16 bit values to memory but it can also save signed 32 bit values when necessary. It always reads signed 16 bit values though.

Mathilda consists of three address generators which produces the addresses for the elements in s_0, s_1 and d respectively, one 16*16 bits multiplier, one 39-bit accumulator, one saturating barrel shifter and one FIFO buffer which holds the intermediate results before writing them down to memory, see figure 32.

The alternative datapath consists of two address generators and four accumulators which operate in parallel, see figure 33.

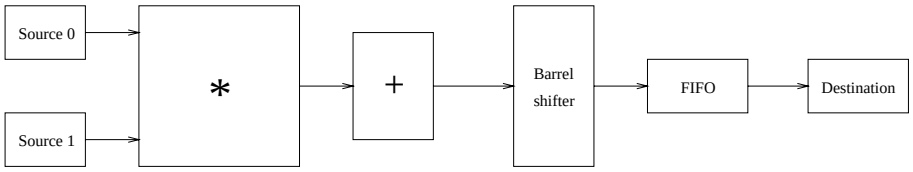


Figure 32: Block diagram of normal Mathilda datapath

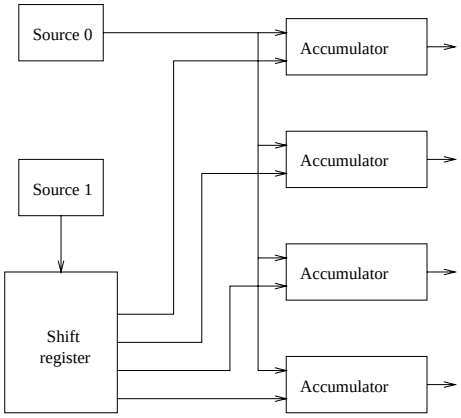


Figure 33: Block diagram of optional Mathilda datapath

The datapaths are controlled by a FSM and two counters corresponding to the i and j indexes, see figure 34.

7.4.1 Address generators and memory access

The address generators generate memory addresses for the different input and output channels. They are all similar and are in fact implemented using a single unit. The address generators are capable of generating addresses to elements in d, s_0 and s_1, see equation 1. This is done using formula 2.

$$adr_{new} = \begin{cases} adr_{old} + stride_0, & i < m-1 \\ adr_{old} + stride_1, & i = m-1 \end{cases} \tag{2}$$

where adr_{old} is the previous element, adr_{new} is the address to the next element, $stride_0$ is the space, in addresses, between two rows and finally $stride_1$ is the space, in addresses, between two columns.

The address generator destination, d, use both $stride_0$ and $stride_1$ even though it really only needs $stride_0$. When saving 32 bit words however you need to specify where the second, least significant, word of the data is to be saved. This is done by using formula 3.

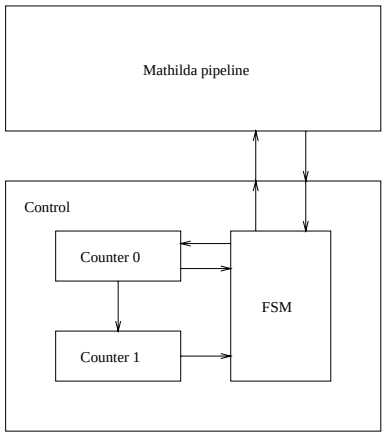


Figure 34: Sketch of Mathilda's controller

$$adr_{new} = \begin{cases} adr_{old} + stride_0, & when\ saving\ the\ most\ significant\ word \\ adr_{old} + stride_1, & when\ saving\ the\ least\ significant\ word \end{cases} \quad (3)$$

where adr_{old}, adr_{new} is as in formula 2, $stride_0$ is the space, in addresses, between the most and least significant word and finally $stride_1$ is the space, in addresses, between the previous least significant word and the next most significant word.

Mathilda accesses memory through the bus arbiter and acts like three DMA channels corresponding to s_0, s_1 and d. The two source channels are controlled so that they never interfere. The destination channel however can ask for memory at the same time as one of the sources and has in those cases higher priority.

7.4.2 Multiplier

The multiplier is the most critical part of Mathilda. Improperly implemented it can be both large and slow. It is vital that the multiplier is fast, and therefore a pipelined design was chosen. A naive pipelined design would have had 16 stages which would have been much too large. Instead a radix-4 Booth coded design was used which can process two bits simultaneously, see chapter 5 in [Nils92]. Since there is only one memory from which the multiplier can get operands, it can only get at most one operand per clock cycle. Thus the multiplier can only produce one result every two clock cycles. This has been used to combine the pipeline stages in pair. The multiplier thus has four stages where each stage is a two cycle 4*16 bit radix-4 Booth multiplier, see figure 35. For further information regarding multipliers see appendix A in [ComA].

Figure 35: Actual BBDS screen dump of Mathilda's multiplier

7.4.3 Accumulator, Barrel shifter and FIFO-buffer

The accumulator is 39 bit wide and has no saturation logic because it would have made the accumulator more complex and slow. The accumulator adds the result in two cycles which is possible due to the multi cycle nature of the multiplier, see section 7.4.2. In the first cycle the 11 least significant bits of the multiplier's result are added to the accumulator and in the second cycle the rest of the result, ie 28 bits.

The barrel shifter corresponds to the 2^k factor of equation 1. It is pipelined and has saturation logic. During the first stage the contents of the accumulator is shifted up to 24 bits up or down and in the second stage overflow is checked. If an overflow is generated the result is saturated using formula 4. Note that the saturated result is a signed 32 bit integer.

$$result_{sat} = \begin{cases} -\$80000000, & result < -\$80000000 \\ result, & -\$80000000 \leq result \leq \$7fffffff \\ \$7fffffff, & result > \$7fffffff \end{cases} \quad (4)$$

Between the barrel shifter and memory is a 5 entry FIFO buffer. The buffer is needed because if Mathilda cannot obtain the bus, it cannot store the results that might be in the previous pipeline stages. These results are instead saved in the FIFO and saved when Mathilda obtains the bus. Another solution would have been to stall the entire pipeline, but since the pipeline is very deep and the pipeline registers are about 32 bits wide, this is not a viable solution.

The FIFO is 32 bit wide and the values are right shifted 16 bits when saved as a 16 bit signed integer. When saved as a 32 bit integer the 16 most significant bits are saved first and then the least 16 bits.

7.4.4 Speedup of stochastic code book search

The single most compute intensive operation in CELP is the stochastic code book search. A 60 samples long vector, from now on called the test sample, has to be compared with 512 code book vectors each 60 samples long. Without any modification of Mathilda the search would have used about 45% of the total availabel time used by CELP. Clearly this operation is a good candidate to be boosted by some kind of dedicated hardware.

The search is performed using 512 correlations between the test sample and each code book-vector. Luckily the stochastic code book uses ternary values, ie the code book-vectors only consist of 0 and ± 1. Thus the correlation resolves to simple additions and subtractions, see formula 5.

$$result_{sat} = \sum_{i=0}^{59} \begin{cases} a_i, & codebookvalue = +1 \\ -a_i, & codebookvalue = -1 \\ 0, & otherwise \end{cases} \quad (5)$$

Though this simplification is simple to implement it gives no speedup! However, the code book can be rearranged to make it possible to run four (4) correlations concurrently. Two code book vectors overlap with only two samples, ie $codebookvector_0(n) = codebookvector_1(n-2)$. Each sample in the code book vectors can be packed into 2 bits and thus 8 samples into one 16-bit word. Thus you can get samples from 4 consecutive vectors if you fetch a 16-bit word of packed code book data. This has been used to make dedicated hardware that performs 4 operations in parallel, see figure 33.

The optional datapath can also compare the four correlations and remember the index of the code book vector with the largest absolute correlation and the value of that correlation. It is thus capable of performing an entire stochastic code book search in one operation.

This datapath operates at twice the speed of the normal datapath since it only

needs to fetch one operand from memory for each parallel operation, not counting the code book values which is fetched once every eighth clock cycle. Thus the ideal speedup is close to 8. However since some additional startup and compare cycles are needed the true speedup is 6 reducing the time spent on stochastic code book search to about 10 % of the total CELP time giving an overall speedup of 1.6.

Perhaps the most elegant feature of this speedup hardware is that it is possible to reuse exactly the same hardware that is used in the multiplier of the normal datapath. The four adders in the multiplier needed only to be widend with 4 bits. The only added hardware are some flipflops, multiplexers and some extra states in the control FSM.

7.4.5 Testing

The digital parts have been simulated by running programs on Dundee that use the different parts of Darwin. When testing Mathilda, test vectors, ie memory dumps, have been taken from the software model before and after an operation. This makes it possible to load an entire memory image into the clock cycle true model, run an operation in hardware and compare the results with those of the verified software model. This has been a very powerful way to test Mathilda for errors. As an additional advantage all test programs written can be used for testing of the fabricated chip. We believe that this approach gives a design that is both easier to verify and contains less errors.

8 Conclusions and summary

A mixed mode ASIC, Darwin, has been developed capable of coding CELP in real time if it is running at 14 MHz. Darwin can however be run at 50 MHz and has a core size of about 30 mm^2. All digital parts have been written entirely in VHDL and have been implemented using a 0.8μm 3.3V process from AMS.

Among other things Darwin consists of a new kind of stand alone DSP unit called Mathilda, and a RISC processor called Dundee. All parts of Darwin have been newly developed and implemented.

The CELP algorithm has been mapped to Mathilda. Dedicated hardware which speedup CELP's stochastic code book searches by a factor of 6 has also been invented and integrated into Mathilda. Using this hardware CELP is speeded up by 60%.

A future improvement would be to integrate RAM for Mathilda on chip. This would enable us to remove one memory port. To lower the power consumption internal caches could be added to the processor and an over clocked scheme could be used. This would reduce the power consumption of the pads in the memory port(s) and help lowering EMI.

The thesis has been a large one and we regretfully conclude that we would have made a better job if we had omitted the analog part and concentrated on the digital part. Our time plan worked very well in the first half of the project and allmost all milestones where reached on-time. But due to difficulties with the Cadence tools the rest of the project where delayed and the prototype production where split into two tape outs.

Furthermore our contact with our assigner, Business Security AB, would have been better if we had worked there and not at the department. This could not be arranged because they had no ASIC tools or suitable computers.

References

[BBDS] Björn Breidegard and Per Andersson, "BBDS - A design tool for architectural evaluation and rapid prototyping of performance critical digital systems", Department of Computer Engineering, Lund University.

[CaTW90] J.P.Campbell, Jr.,T.E.Tremain,V.C.Welch, "The proposed Federal Standard 1016 4800 bps voice coder: CELP", Speech Technology, pp.58-64, Apr./May 1990

[ComA] J. L. Hennessy, D. A. Patterson, "Computer architecture, A quantitative approach",Second edition Morgan Kaufmann Publishers Inc., 1996

[CSDM] Lars Philipson, "Computer System Design Methodology" a course given at Department of Computer Engineering, Lund University, lars@dit.lth.se, www.dit.lth.se

[CYX] Austria Mikro International AG, "2.0-Micron, 1.2-Micron, 1.0-Micron and 0.8-Micron Standard Cell Databook", Austria, 1995.

[DodCELP] CELP example program, CELP Version 3.2 C code

[Juhl] Anders Juhlin,"Parallel step-wise refinement" (Den sneda vågens metod) LEWE-project, Department of Informatics, Lund University (Swedish)

[Lang94] A.Langi, W.Grieder, W.Kinsner "Fast CELP algorithm and implementation for speech compression", IEEE 1994 proc. Digital Communications Conference, The University of Manitoba, Winnipeg, MB, Canada 1994

[m68k] Motorola inc., "M68000 - Family Programmer's Reference Manual", Motorola inc.,1992

[McEl] Ciarán McElroy,"Speech Coding", http://wwwdsp.ucd.ie/speech/tutorial/speech_coding/speech_tut.html.

[Mol94] Bength Arne Molin,"Analog konstruktion i CMOS", Department of Applied Electronics,Lund University, 1994.

[MOSAmp] "MOS operational Amplifier Design - A Tutorial Overview" IEEE J. Solid State Circuits vol. SC-17, no 6, pp. 969-982, Dec 1982

[Nils92] P Nilsson, "A CMOS VLSI Cell Library for Digital Signal Processing", Department of Applied Electronics, Lund University, 1992

[Std1016] Fenichel, R. Federal standard 1016, "Telecommunications: Analog to digital conversion of radio voice by 4,800 bit/second code exited linear prediction (CELP)", FSC TELE, General Services Administration Office Of Information Resources Management. National Communications System, Office of Technology and Standards, Washington, DC 20305-2010, 14 February 1991

[Werner] B. Werner, K. Ranerup, B. Breidegard, G. Jennings and L. Philipson, "Werner Diagrams - Visual Aid for Design of Synchrounous Systems", Department of Computer Engineering, Lund University, 1992.

[Yuan95] Jiren Yuan, "Successive Approximation A/D Converters", Slides 951004, LiTH, Linköping, Sweden.

[Zej95] Rifat Zejnic, "CADENCE 4.3 En sammanfattning", Department of Applied Electronics, Lund University, 1995

[Öwa93] V.Öwall, P.Andreani, L.Brange, P.Nilsson, A.Wass and M.Torkelson "Custom DSP Design of a GSM Speech Coder",Journal of VLSI Signal Processing, pp. 213-228, November 1995

A Dundee instructionset

This section is a small summary of Dundee's instruction set. It is not a complete manual in writing programs for Dundee

There are 16 general registers denoted r0 to rf. Registers can also be clustered into pairs when an address is calculated. All integers are signed unless otherwise stated.

A.1 Load and store instructions

Syntax	*Description*
ldi rx,const(ry)	Load a 16-bit word from memory at address ry (as a register pair) + const (16 bit constant) into register rx.
ldq rx,const(ry)	Load a 16-bit word from memory at address ry (as a register pair) + const (unsigned 4 bit constant) into register rx.
sti const(ry),rx	Store rx in memory at address ry (as a register pair) + const (16 bit constant).
stq const(ry),rx	Store rx in memory at address ry (as a register pair) + const (unsigned 4 bit constant).
lda rx,const(ry)	Load a 16-bit word from alternative memory at address ry (as a register pair) + const (unsigned 4 bit constant) into register rx.
sta const(ry),rx	Store rx in alternative memory at address ry (as a register pair) + const (unsigned 4 bit constant).
romld rx,(ry)	Load a 16-bit word from rom at address ry (as a register pair) into register rx.

A.2 Constant add instructions

Syntax	*Description*
addi rx,ry,#const	Add ry and const (16 bit integer) and store into rx. Affects flags.
addq rx,ry,#const	Add ry and const (4 bit integer) and store into rx. Affects flags.
addni rx,ry,#const	Add ry and const (16 bit integer) and store into rx.
addnq rx,ry,#const	Add ry and const (4 bit integer) and store into rx.

A.3 Normal addition and subtraction instructions

Syntax	*Description*
add rx,ry,rz	Add ry and rz and store into rx. Affects flags.
sub rx,ry,rz	Subtract rz from and store into rx. Affects flags.

A.4 Basic logic and arithmetic instructions

Syntax	*Description*
or rx,ry	rx logically ored with ry and stored in rx. Affects flags.
and rx,ry	rx logically anded with ry and stored in rx. Affects flags.
not rx,ry	ry logically noted and stored in rx. Affects flags.
xor rx,ry	rx logically xored with ry and stored in rx. Affects flags.
neg rx,ry	ry is negated and stored in rx. Affects flags.
sext rx,ry	the lowest 8 bits of ry is signed extended into 16 bits and stored in rx. Affects flags.
cmpq rx,#const	Compare rx with const (4 bit integer). Affects flags.
cmp rx,ry	Compare rx with ry. Affects flags.
cmpm rx,ry	Compare rx with ry. Used for multiprecision arithmetic. Affects flags.
addc rx,ry	Add rx with ry taking carry in account and store in rx. Used for multiprecision arithmetic. Affects flags.
subc rx,ry	Subtract ry from rx taking carry in account and store in rx. Used for multiprecision arithmetic. Affects flags.
movei rx,#const	Move const (16 bit integer) into rx.
moveq rx,#const	Move const (16 bit integer) into rx.
move rx,ry	Copy ry to rx.
nop	Do nothing.
lsl rx,#const	rx is logically shifted left const steps. Const can be 1,2,4 or 8. Affects flags.
lsr rx,#const	rx is logically shifted right const steps. Const can be 1,2,4 or 8. Affects flags.
asl rx,#const	rx is arithmetically shifted left const steps. Const can be 1,2,4 or 8. Affects flags.
asr rx,#const	rx is arithmetically shifted right const steps. Const can be 1,2,4 or 8. Affects flags.
rol rx,#const	rx is rotated left const steps. Const can be 1,2,4 or 8. Affects flags.
ror rx,#const	rx is rotated right const steps. Const can be 1,2,4 or 8. Affects flags.

A.5 Misc instructions

<table>
<tr><th>Syntax</th><th>Description</th></tr>
<tr><td>muls rx,ry</td><td>ry is multiplied with rx. The result is stored in special register m0 and m1.</td></tr>
<tr><td>divu rx,ry</td><td>rx is divided with ry. The result is stored in special register d0 and d1.</td></tr>
<tr><td>mts sx,ry</td><td>Copy ry to special register sx. sx can be any of:

<table>
<tr><th>Register name</th><th>Description</th></tr>
<tr><td>ip</td><td>interrupt pending register</td></tr>
<tr><td>sr</td><td>status register</td></tr>
<tr><td>ib</td><td>interrupt handler base address</td></tr>
<tr><td>im</td><td>interrupt mask</td></tr>
<tr><td>t0</td><td>temporary register 0</td></tr>
<tr><td>t1</td><td>temporary register 1</td></tr>
<tr><td>t2</td><td>temporary register 2</td></tr>
<tr><td>t3</td><td>temporary register 3</td></tr>
<tr><td>ph</td><td>high 16 bits of program counter</td></tr>
</table></td></tr>
<tr><td>mfs rx,sx</td><td>Copy special register sx to register rx. sx can be any of:

<table>
<tr><th>Register name</th><th>Description</th></tr>
<tr><td>ip</td><td>interrupt pending register</td></tr>
<tr><td>sr</td><td>status register</td></tr>
<tr><td>vr</td><td>hardware version register</td></tr>
<tr><td>pc</td><td>program counter</td></tr>
<tr><td>t0</td><td>temporary register 0</td></tr>
<tr><td>t1</td><td>temporary register 1</td></tr>
<tr><td>t2</td><td>temporary register 2</td></tr>
<tr><td>t3</td><td>temporary register 3</td></tr>
<tr><td>d0</td><td>quotient of the division</td></tr>
<tr><td>d1</td><td>remainder of the division</td></tr>
<tr><td>m0</td><td>low 16 bits of the result from a multiply</td></tr>
<tr><td>m1</td><td>high 16 bits of the result from a multiply</td></tr>
<tr><td>ph</td><td>high 16 bits of program counter</td></tr>
</table></td></tr>
</table>

A.6 Control flow instructions

Syntax	*Description*
sint	Do a software interrupt.
rint	Return from interrupt.
jr (rx)	Jump to adress contained in register pair rx.
bccs target	Branch to target if the condition cc is true, see table below. The target must be within 128 words from the branch instruction. One branch delay slot is used.
bccl target	Branch to target if the condition cc is true, see table below. The target must be within 512 kwords from the branch instruction. One branch delay slot is used.

The conditions are:

Name	*Description*
cc	carry clear
ls	lower or same
cs	carry set
lt	less than
eq	equal
mi	minus
ne	not equal
ge	greater or equal
pl	plus
gt	greater than
hi	higher
vc	overflow clear
ht	higher than
vs	overflow set
ra	always true

B Introduction to speech coding

Most of the information presented in this section was found in a more detailed introduction to speech coding by Ciarán McElroy [McEl].

B.1 Human speech

When people talk, sound is generated by blowing air through the vocal tract. The sound building up the speech can be divided into two groups, voiced and unvoiced. Lungs are used as air supply and the sound is formed when the air passes the vocal folds which are two membranes located in the larynx. During voiced speech the air flow causes the folds to vibrate with a frequency determined by the length of the folds and their tension. This pitch frequency is in the range of 50 - 400 Hz with different mean depending on person and sex. Unvoiced sound is created when the vocal folds are open and the air passes freely into the rest of the vocal tract. Noise like sound is generated by constricting the vocal tract close to the lips.

The size and the form of the vocal tract acts as a spectral sharping filter of the voiced and unvoiced speech signal and therefore changes the spectrum of the sound. The shape and size of the vocal tract is changed with movement of the tongue and mouth.

The nasal cavity is acoustical coupled to the tract when the velum is lowered and changes the nature of sound dramatically.

B.2 The speech signal

Speech signal uses a bandwidth of approximately 7 kHz and a spectral plot of voiced speech (figure 36) shows a low-pass nature. In the spectrum a number of resonance tops called formants are shown. In the time domain the signal has a clear periodic look.

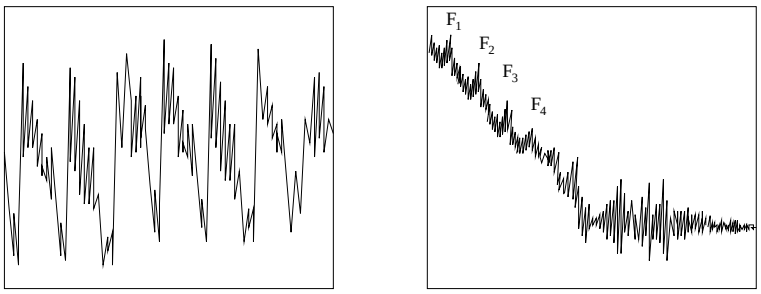


Figure 36: Time and frequency of voiced speech

The unvoiced spectrum in figure 37 is flat or high-pass and no formants and pitch structure are visible. The signal looks random and has lower energy than the voiced speech.

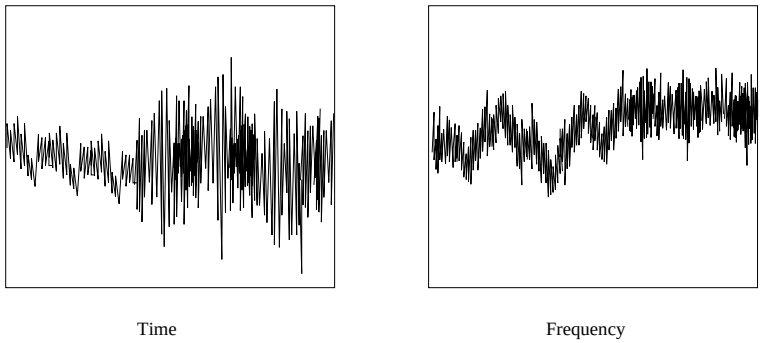


Figure 37: Time and frequency of unvoiced speech

B.3 Speech receiver

The ear is the receiver of the speech signal after transportation by air. Sound is waves of air pressure changes traveling in a three dimensional medium. The ear is quite complex and may simplified be viewed as a very good logarithmic microphone. Some characteristics about the ear and the way humans recognize sound is important to speech coding. The ear has limited sensitivity which varys with frequency. If sound is weaker than the threshold of audibility no detection will occur. The phenomenon when the threshold is increased for a given frequency if a larger signal exists nearby is called masking and is widely used in all types of sound coding. The formants are important to mask unwanted noise inaudible to the listener.

B.4 Sampling and quantization

Before a natural signal may be processed by a computer system the signal must be sampled and quantized.

B.4.1 Sampling

Sampling is the process where a timecontinous signal is converted to a discrete time signal by measuring the continuous signal at periodic intervals. If the number of measurements (samples) per second increases, the sampled signal is a better approximation of the original signal. From a sampled signal the original signal may be perfectly reconstructed if the original signal only contains frequency components less then the Nyquist frequency.

B.4.2 Quantization

The approximation of a continuous-valued measurement to a discrete-valued measurement is called quantization and is done by a quantisizer. The quantized output is chosen from a finite set of values and the difference between the input and the output is called the quantization error.

Quantization may be done with several techniques depending on the nature of the signal. Linear quantization uses equal distance between all levels and does not make any assumptions about the nature of the signal. Toll quality telephone speech is possible to obtain with 13 bit linear quantisizer.

A logarithmic quantisizer sends the signal through a compressor with logarithmic characteristics before or after quantization and therefore the distance between quantization level differs depending on the signal level. Small distance for small signals and large distance for large signals. Two popular logarithmic compressors are A-law and μ-law. Using these schemes toll quality speech can be obtained with a 8 bit logarithmic quantisizer.

If the signal has special characteristics, other custom nonuniform quantization can be used to minimize the quantization error by choosing the smallest quantization levels where the signal is most likely to be or where the signal is most critical.

Vector quantization is used when more than one value is quantized together in a block of N samples. The sample is a point in a N-dimensional space. Vector quantization is better than scalar quantization but is more complex.

B.5 Waveform coder

A waveform coder sends information about the waveform to the receiver which tries to restore the original signal. The waveform coder makes no assumptions about the signal to be coded.

B.5.1 Time domain coding

Pulse Code Modulation, PCM, is a CODEC that just sends the sampled and quantized signal values directly to the receiver.

Differential Pulse Code Modulation, DPCM, sends the difference between adjacent samples to the receiver. If the signal has high correlation between samples this reduces the bit rate.

Adaptive Differential Pulse Code Modulation, ADPCM, uses a scheme to change quantiziser and predictor according to changes in the speech signal. The predictor coefficients for a speech block are calculated and transmitted to the receiver. This method further decreases the information to be send but is more sensitive to bit errors.

B.5.2 Frequency domain coding

The coder splits the signal in different blocks containing different frequency components. Each block may then be treated independently in an optimal way.

A sub-band coder uses different filters to divide the input signal into different frequency bands which are frequency shifted and low pass filtered. Each band is then treated as a new waveform and is coded using one of the time domain techniques above. The receiver restores each subband and modulates it back to the original position.

In a transform coder a segment of the signal is windowed and transformed to a selected domain. Different transform coefficients are then assigned different number of quantization bits depending on importance. The receiver does the inverse transform and the original signal is reconstructed. Transforms that may be used are the Discrete Fourier transform, DFT, and the Karhunen Loeve Transform, KLT. This technique is not normally used in speech coding because of its complexity but it is used when general sound is coded. The MPEG sound substandard is built around transform coding.

B.6 Voice coding

If speech is the only signal, a good voice coder could be realized if some properties about the sound that build up the speech are used. Voice coders use a simplified model of the speech generation, see figure 38.

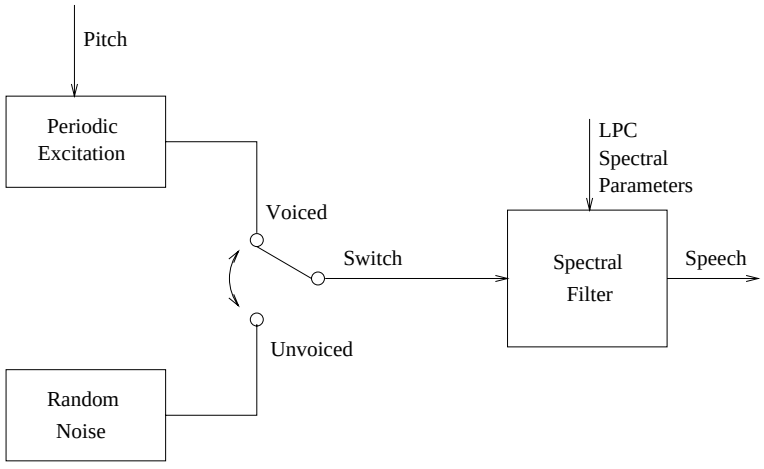


Figure 38: Speech generation

If the speech is voiced, excitation is done with a series of impulses with a distance equal to the pitch period. When the speech is unvoiced, the excitation is random noise. The output spectrum formed in the vocal tract is created with a spectral filter.

All parameters in the speech creation model are determined in the speech analyzer. This can be done in many ways and this part is often the difference between coders. Those parameters are transmitted over the channel to the synthesizer which uses the speech parameters to create a waveform that sounds similar to the original signal. With this type of coder quite low bit rate may be used to transfer toll quality speech.

B.6.1 Channel vocoder

The ear is very insensitive to phase and therefore the vocoder in figure 39 is capable of providing highly intelligible speech at bit rates in the region of 2.4 kb/s.

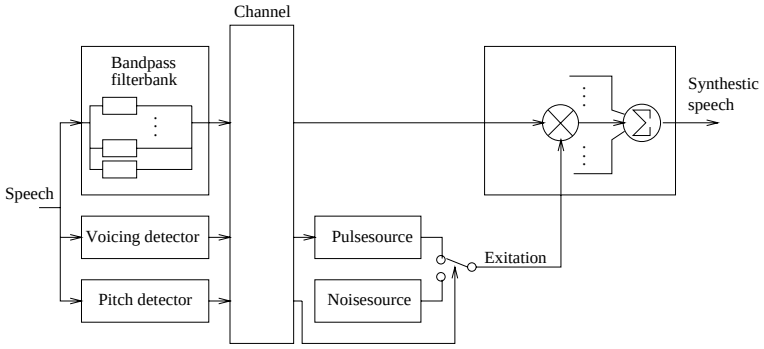


Figure 39: Vocoder

The speech is divided into frames of about 20 ms which are analyzed in a filter bank to obtain a voice spectrum. This spectrum, the pitch and an indication for unvoiced sound are transmitted to the receiver. The receiver synthesizes sound for each sub-band in the spectrum with pitch or noise excitation and finally adds everything together to create speech.

B.6.2 Formant voice coder

A large part of the information in voiced speech signal is contained in the formants. A bit rate less than 1 kb/s is possible to achieve by only transmitting information of the formants, however the formants are very difficult to determine and therefore this type of voice coder is not used.

B.6.3 Linear predictive voice coder

The Linear predictive voice coder, LPC, assumes that an all pole infinite impulse response filter, see equation 6 can describe the vocal tract. Each speech sample is predicted to be a linear combination of the previous samples.

$$H(z) = \frac{G}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_p z^{-p}} \tag{6}$$

A frame of speech is first analyzed to determine which filter parameters that minimize the prediction error. The frame is then passed through the filter to remove the sample to sample correlation. The pitch is then more easy to find because the long term correlation is more visible. This model of the vocal tract works quite well and is used to create highly intelligible speech at 2.4 kb/s.

B.7 Hybrid coding

The most successful and commonly used hybrid coder is the time domain Analysis by Synthesis, AbS, coder. It attempts to fill the gap between waveform and source coders as waveform coders may produce speech at bit rates down to about 16 kb/s and source coders can provide intelligible speech around 2.4 kb/s but with the natural sound of the speech missing.

Hybrid coders often use the filter model of the LPC voice coder but a better excitation signal is used. The signal is selected so the synthesized sound matches the original sound as closely as possible.

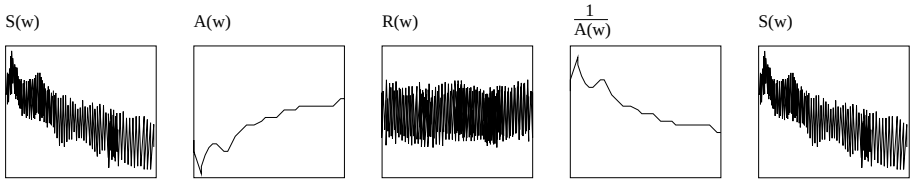


Figure 40: Speech signal, filtering, noise, inverse filtering, speech

B.7.1 Pulse coder

The first hybrid coder was presented 1982 by Atal and Remde and was a Multi-Pulse Excited, MPE, coder. In the MPE-coder the excitation is a number of non-zero pulses with individual position and amplitude.

In the Regular-Pulse Excited, RPE, coder the excitation consists of regular spaced pulses with same amplitude. A simplified RPE coder is used in the pan-European GSM mobile telephone system at 13 kb/s [Öwa93].

B.7.2 Residual coder

The Residual Excited Linear Prediction, RELP, coder uses a small portion of whats left after the LPC filter has removed the short time correlation as excitation, figure 40. The residual is almost flat and uniform over the frequency-band so only a small portion of the residual is needed.

The residual is passed through a low pass filer (1kHz) and then waveform coded. The receiver reconstructs the residual up to 1kHz and duplicates it over the frequency-band. The drawback of the RELP coder is that the pitch frequency above 1kHz is lost. RELP coders may be used around 9,6 kb/s.

B.7.3 Code book coder

Code book Excited Linear Prediction, CELP, coder includes a LPC vocal tract predictor and a pitch predictor. The residual after the pitch predictor is almost random noise and the CELP coder vector-quantisizes and transmits this noise to the receiver. The coder includes a code book containing vectors, ie samples, of Gaussian noise. For every frame the CELP algorithm performs an exhaustive analysis-by-synthesis search through every vector in the code book. The index and gain of the vector which is the best perceptual match to the residual are transmitted. This means a 15 bit excitation [Lang94] compared to a 47-bit excitation which GSM has[Öwa93].

The drawback of the algorithm is the massive calculations that must be done for every frame. But some simplifications may be done and a lot of more or less computer intensive schemes have been developed. The high computional speed reachable with state of the art DSP's have made it possible to develop real time CELP coders. CELP coding has been quite successful in producing toll quality speech communications at rates between 4.8kb/s and up to 16kb/s. A 16kb/s coder produces speech which is almost indistinguishable from 64 kb/s PCM coded speech.

A short introduction to the celp algorithm can be found below:

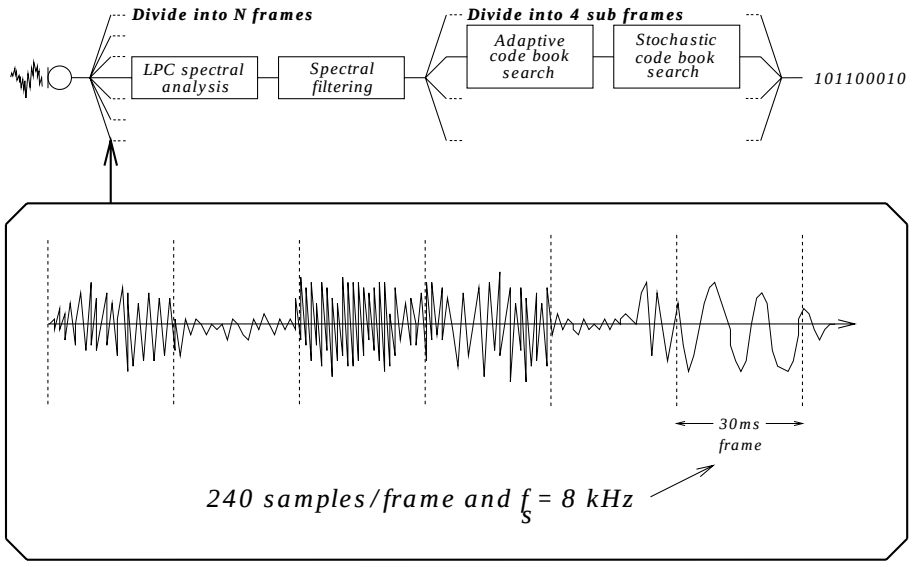


Figure 41: First the sampled sound is divided into N frames

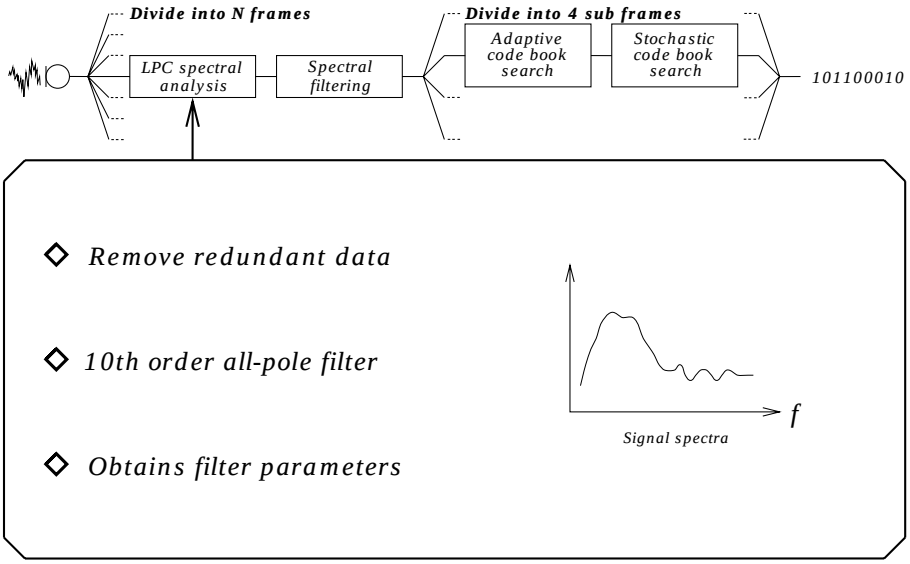


Figure 42: Then filter parameters for a 10th order all-pole filter are obtained

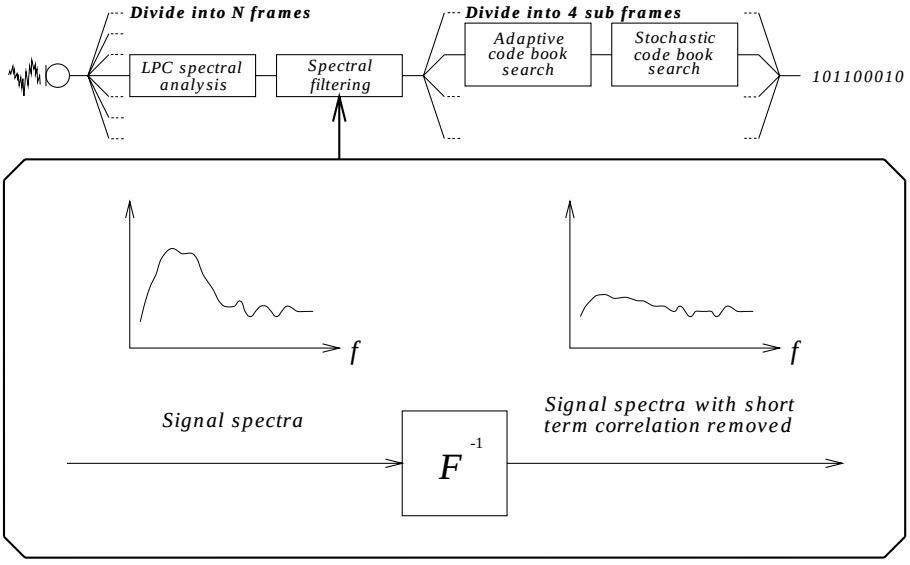


Figure 43: This filter is then used to remove redundant data from the speech

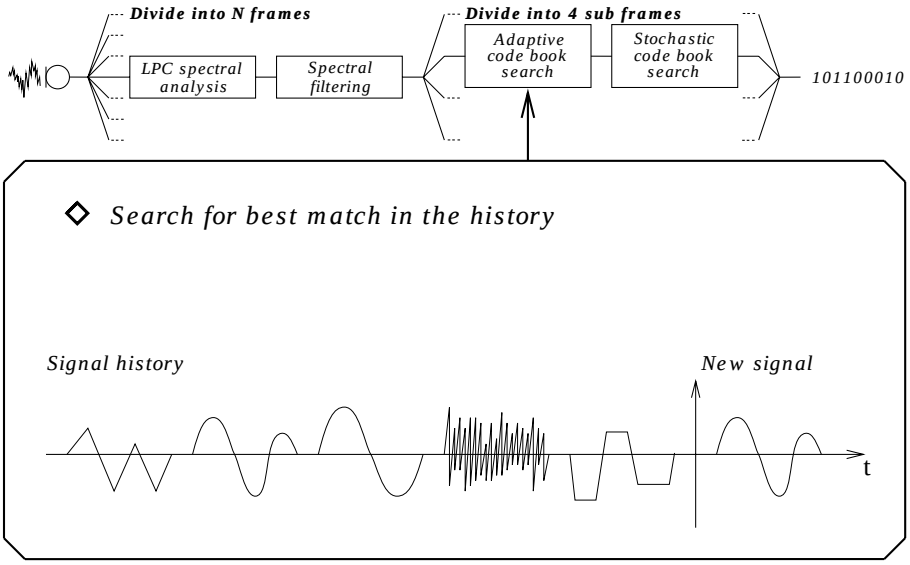


Figure 44: The adaptive code book is searched for the best match in the history which is stored in a circular buffer

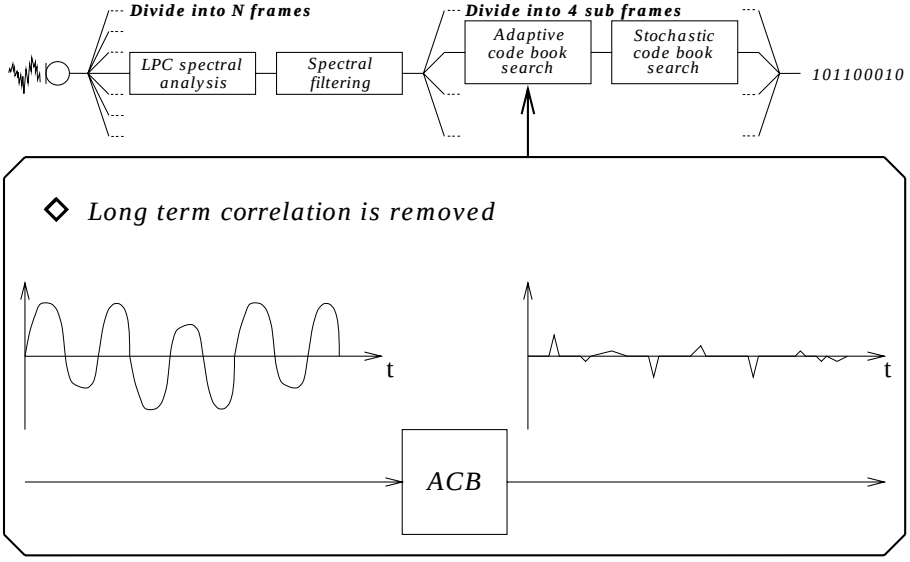


Figure 45: The history is removed and thuns the long term correlation

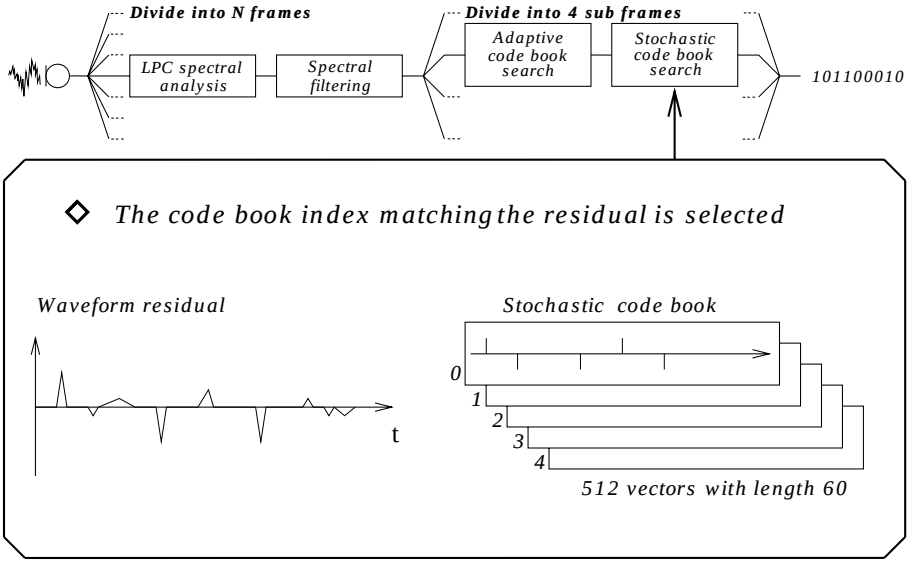


Figure 46: Finally, the residual is matched against the noise in the stochastic code book

B.7.4 Multi band code book coder

The CELP coder may be improved and one way of doing this is to split the excitation into different sub-bands. The Multi band Excitation, MBE, coder splits the residual spectrum into a number of signals with less bandwidth which is individually vector quantized. This results in a coder that simultaneously outputs voiced and unvoiced sound just as in real human speech.

C Analog design

This appendix contains notes and calculations regarding analog blocks in the design. All calculations uses process constants found in table 3.

Parameter	Value	Unit
ϕ_p	1	V
ϕ_n	1	V
γ_p	0,45	$\sqrt{V}$
γ_n	0,7	$\sqrt{V}$
k_p	30	$\mu A/V^2$
k_n	70	$\mu A/V^2$
V_{T_p}	0,75	V
V_{T_n}	0,88	V
λ	0,05	$\frac{1}{V}$
C_{oxide}	2,2	$fF/\mu m^2$
$C_{metal1-poly}area$	0,053	$fF/\mu m^2$
$C_{metal1-poly}perimeter$	0,051	$fF/\mu m$

Table 3: Process parameters $0.8\mu m$ CMOS

C.1 Amplifier buff_small

The buffer amplifier does not need to be fast but needs to drive current to the outside world through the output pad. Worst case load is estimated to be $1.5pF$ and is charged with the amplifier in a follower configuration. Bandwidth on the input signal is less than 16kHz but the op-amp bandwidth is chosen to 0.25MHz and the slew rate is calculated to

$$\frac{3.3V}{62.5\mu s} = 0.05V/\mu s$$

and selected 10 times larger $S_R = 0.5V/\mu s$

Specifications:

$M = 3 \Rightarrow \phi_m = 90 - arctan\frac{1}{M} \approx 72^\circ$

where M is the distance between G_B and pole number 2.

$G_B = 2\pi \cdot 0.25 \cdot 10^6 rad/s$

$S_R = 0.5V/\mu s$

$C_L = 1.5pF$

$V_{DD} = 3.3V$

The op-amp must be compensated and C_C was chosen to $0.8 \cdot C_L = 1.2pF$.

With this capacitance a $C_{parasit_1}$ of around $0.1pF$ appears in the design and may be ignored because it is small.

1. Current in input and output stages:

$$I_1 = S_R \cdot [C_C + C_{parasit_1}]$$

$$I_2 = S_R \cdot [C_C + C_L]$$

and calculated to $I_1 = 0.65\mu A$ and $I_2 = 1.35\mu A$.

2. For the input stage g_{m_i} can be expressed

$$g_{m_i} = G_B \cdot C_C$$

and because

$$g_{m_i} = \sqrt{\frac{K}{2}\left(\frac{W}{L}\right)_i \frac{I_1}{2}}$$

the relationship can be obtained as

$$\left(\frac{W}{L}\right)_i = \frac{G_B^2 C_C^2}{I_1 k_p} = \frac{(2\pi \cdot 0.25 \cdot 10^6)^2 (1.2 \cdot 10^{-12})^2}{0.65 \cdot 10^{-6} \cdot 30 \cdot 10^{-6}} = 0.18$$

3. The output transistor ratio can be calculated based upon

$$pole_2 = \frac{1}{-sp_2} = \tau_2 = \frac{C_L}{g_{m_6}}$$

$$M \cdot G_B = -sp_2 \approx \frac{g_{m_6}}{C_L}$$

$$g_{m_6} = M \cdot G_B \cdot C_L = 3 \cdot 2\pi \cdot 0.25 \cdot 10^6 \cdot 1.5 \cdot 10^{-12} = 7.1 \cdot 10^{-6}$$

$$\left(\frac{W}{L}\right)_6 = \frac{g_{m_b}^2}{2k_n I_2} = 0.27$$

4. $V_{bias} = V_{GS} - V_T$ is selected to 0.4. This selection gives small offset and better output swing but decreases the accuracy in A_0 when small changes in V_T change the gain dramatical.

$$\left(\frac{W}{L}\right)_{5,7} = \frac{I}{\frac{k_p}{2}(V_{GS} - V_T)^2}$$

$$\left(\frac{W}{L}\right)_5 = \frac{I_1}{\frac{k_p}{2}(0.4)^2} = 0.27$$

$$\left(\frac{W}{L}\right)_7 = \frac{I_2}{\frac{k_p}{2}(0.4)^2} = 0.56$$

5. More bias

$$V_{DS_3} = V_{GS_3} = V_{GS_4} = V_{DS_4} = V_{GS_6}$$

due to same current and same dimensions

$$\left(\frac{W}{L}\right)_3 = \left(\frac{W}{L}\right)_4 = \left(\frac{W}{L}\right)_6 \frac{I_1/2}{I_2} = 0.24$$

6. The nulling resistor is implemented as a transistor $M8$ in the linear region. Ie the zero is placed above the third pole which is canceled out.

$$g_d = \left(\frac{\partial I_D}{\partial V_{DS}}\right)^\circ = k_n \frac{W}{L}(V_{GS}^\circ - V_T) =$$

$$k_n \left(\frac{W}{L}\right)_8 \left(V_{GS}^\circ - V_{TO} - \gamma\left(\sqrt{\phi_B - V_{BS}} - \sqrt{\phi_B}\right)\right)$$

Voltage over $M4$ will be

$$\frac{I_1}{2} = \frac{k_n}{2}\left(\frac{W}{L}\right)_4 (V_{GS}^2 - V_T)^2$$

$$V_{BS} = \sqrt{\frac{I_2}{k_n\left(\frac{W}{L}\right)_4}} + V_{T_n}$$

$$= \sqrt{\frac{1.35 \cdot 10^{-6}}{70 \cdot 10^{-6} \cdot 0.24}} + 0.88 = 1.16$$

$$V_{GS_8} = V_{BS_8} = 3.3 - 1.24 = 2.14$$

The zero vanishes when

$$R_Z = \frac{1}{g_{d_8}} = \frac{1}{g_{m_6}}$$

$$\left(\frac{W}{L}\right)_8 = \frac{g_{m_6}}{k_n\left(V_{GS}^\circ - V_T\right)} =$$

$$\frac{g_{m_6}}{k_n\left(2.14 - 0.88 - 0.7 \cdot \left(\sqrt{1 + 2.14} - \sqrt{1}\right)\right)} = 0.141$$

If minimal dimension is selected to 4μ all transistor ratios can be found in table 4.

$$
\begin{array}{lcl}
\left(\frac{W}{L}\right)_i = 0.5 & \Rightarrow & \left(\frac{W}{L}\right)_i = \frac{8}{16} \\
\left(\frac{W}{L}\right)_3 = \left(\frac{W}{L}\right)_4 = 0.25 & \Rightarrow & \left(\frac{W}{L}\right)_{3,4} = \frac{4}{16} \\
\left(\frac{W}{L}\right)_6 = 1 & \Rightarrow & \left(\frac{W}{L}\right)_6 = \frac{4}{4} \\
\left(\frac{W}{L}\right)_5 = 0.17 & \Rightarrow & \left(\frac{W}{L}\right)_5 = \frac{4}{24} \\
\left(\frac{W}{L}\right)_7 = 0.56 & \Rightarrow & \left(\frac{W}{L}\right)_7 = \frac{4}{7,1} \\
\left(\frac{W}{L}\right)_8 = 0.08 & \Rightarrow & \left(\frac{W}{L}\right)_8 = \frac{2}{24}
\end{array}
$$

Table 4: Transistor ratios

Some simulation tuning of the design has also been done. Optimization of the compensation circuit gives $C_C = 1.0pF$ and $\left(\frac{W}{L}\right)_8 = \frac{2}{24}$.

Simulation results for buff_small:

$A_o = 224K \rightarrow 107dB$

Unity gain $= 342k$

Gain margin $= 87^o$

Input offset $= -145\mu V$ (V_{in} = 1.5V DC)

The amplifier can handle extra load up to 3.0pF with full range signal and 10.0pF with 1 V signal.

C.2 Amplifier buff_fast

This buffer amplifier must be faster and capable to charge the sample and hold capacitance. Worst case load is estimated to be $2.8pF$ and is charged with the amplifier in follower configuration. Bandwidth on the signal is less than 0.5MHz and the slew rate is calculated to

$$\frac{3.3V}{2.0\mu s \cdot 6cycles} = 0.275V/\mu s$$

and selected 10 times larger $S_R = 2.75V/\mu s$

Specifications:

$M = 3 \Rightarrow \phi_m = 90 - arctan\frac{1}{M} \approx 72^\circ$

$G_B = 2\pi \cdot 0.5 \cdot 10^6 rad/s$

$S_R = 2.75V/\mu s$

$C_L = 2.8pF$

$V_{DD} = 3.3V$

The op-amp must be compensated and C_C was chosen to $0.8 \cdot C_L = 2.3pF$. With this capacitance a $C_{parasit_1}$ of around $0.2pF$ appears in the design.

1. Current in input and output stages:

$$I_1 = S_R \cdot [C_C + C_{parasit_1}]$$

$$I_2 = S_R \cdot [C_C + C_L]$$

is calculated to $I_1 = 6.9\mu A$ and $I_2 = 14.0\mu A$.

2. For the input stage g_{m_i} can be expressed

$$g_{m_i} = G_B \cdot C_C$$

and because

$$g_{m_i} = \sqrt{\frac{K}{2}\left(\frac{W}{L}\right)_i \frac{I_1}{2}}$$

the relationship between W and L can be obtained as

$$\left(\frac{W}{L}\right)_i = \frac{G_B^2 C_C^2}{I_1 k_p} = \frac{(2\pi \cdot 0.5 \cdot 10^6)^2 (2.3 \cdot 10^{-12})^2}{6.9 \cdot 10^{-6} \cdot 30 \cdot 10^{-6}} = 0.25$$

3. The output transistor ratio can be calculated based upon

$$pole_2 = \frac{1}{-sp_2} = \tau_2 = \frac{C_L}{g_{m_6}}$$

$$M \cdot G_B = -sp_2 \approx \frac{g_{m_6}}{C_L}$$

$$g_{m_6} = M \cdot G_B \cdot C_L = 3 \cdot 2\pi \cdot 0.5 \cdot 10^6 \cdot 2.8 \cdot 10^{-12} = 26.4 \cdot 10^{-6}$$

$$\left(\frac{W}{L}\right)_6 = \frac{g_{m_b}^2}{2k_n I_2} = 0.36$$

4. $V_{bias} = V_{GS} - V_T$ is selected to 0.4. This selection gives small offset and better output swing but decreases the accuracy in A_0 when small changes in V_T change the gain dramatical.

$$\left(\frac{W}{L}\right)_{5,7} = \frac{I}{\frac{k_p}{2}(V_{GS} - V_T)^2}$$

$$\left(\frac{W}{L}\right)_5 = \frac{I_1}{\frac{k_p}{2}(0.4)^2} = 2.88$$

$$\left(\frac{W}{L}\right)_7 = \frac{I_2}{\frac{k_p}{2}(0.4)^2} = 5.84$$

5. More bias

$$V_{DS_3} = V_{GS_3} = V_{GS_4} = V_{DS_4} = V_{GS_6}$$

due to same current and same dimensions

$$\left(\frac{W}{L}\right)_3 = \left(\frac{W}{L}\right)_4 = \left(\frac{W}{L}\right)_6 \frac{I_1/2}{I_2} = 0.25$$

6. The nulling resistor is implemented as a transistor $M8$ in the linear region

$$g_d = \left(\frac{\partial I_D}{\partial V_{DS}}\right)^{\circ} = k_n \frac{W}{L}(V_{GS}^{\circ} - V_T) =$$

$$k_n \left(\frac{W}{L}\right)_8 \left(V_{GS}^{\circ} - V_{TO} - \gamma\left(\sqrt{\phi_B - V_{BS}} - \sqrt{\phi_B}\right)\right)$$

Voltage over $M4$ will be

$$\frac{I_1}{2} = \frac{k_n}{2}\left(\frac{W}{L}\right)_4 (V_{GS}^2 - V_T)^2$$

$$V_{BS} = \sqrt{\frac{I_2}{k_n\left(\frac{W}{L}\right)_4}} + V_{T_n}$$

$$= \sqrt{\frac{14.0 \cdot 10^{-6}}{70 \cdot 10^{-6} \cdot 0.25}} + 0.88 = 1.77$$

$$V_{GS_8} = V_{BS_8} = 3.3 - 1.03 = 1.53$$

The zero vanishes when

$$R_Z = \frac{1}{g_{d_8}} = \frac{1}{g_{m_6}}$$

$$\left(\frac{W}{L}\right)_8 = \frac{g_{m_6}}{k_n\left(V_{GS}^{\circ} - V_T\right)} =$$

$$\frac{g_{m_6}}{k_n\left(1.53 - 0.88 - 0.7 \cdot \left(\sqrt{1 + 1.53} - \sqrt{1}\right)\right)} = 1.59$$

If minimal dimension is selected to 4μ all transistor ratios can be found in table 5.

Tuning of the compensation circuit gives $C_C = 1.5pF$ and $\left(\frac{W}{L}\right)_8 = \frac{2}{4}$.

Simulation results for buff_fast:

$A_o = 42.7K \rightarrow 92dB$

Unity gain $= 1.2M$

Gain margin $= 92^o$

Input offset $= -22.14\mu V$ (V_{in} = 1.5V DC)

This op-amp have been test loaded up to 4.5pF.

$\left(\frac{W}{L}\right)_i = 1$	$\Rightarrow$	$\left(\frac{W}{L}\right)_i = \frac{8}{8}$
$\left(\frac{W}{L}\right)_3 = 0.25\ \left(\frac{W}{L}\right)_4 = 0.25$	$\Rightarrow$	$\left(\frac{W}{L}\right)_{3,4} = \frac{4}{14}$
$\left(\frac{W}{L}\right)_6 = 1.5$	$\Rightarrow$	$\left(\frac{W}{L}\right)_6 = \frac{6}{4}$
$\left(\frac{W}{L}\right)_5 = 2.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_5 = \frac{8}{4}$
$\left(\frac{W}{L}\right)_7 = 5.84$	$\Rightarrow$	$\left(\frac{W}{L}\right)_7 = \frac{23.4}{4}$
$\left(\frac{W}{L}\right)_8 = 0.5$	$\Rightarrow$	$\left(\frac{W}{L}\right)_8 = \frac{2}{4}$

Table 5: Transistor ratios

C.3 Comparator

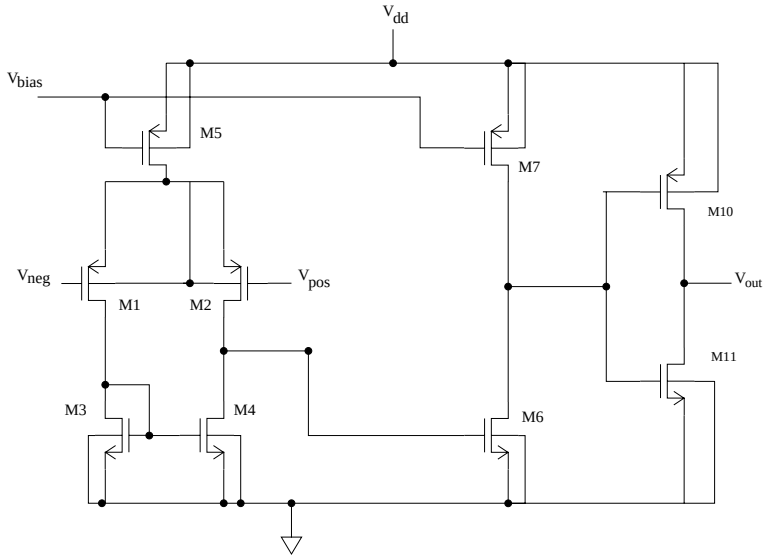


Figure 47: Comparator schematic

The comparator is an uncompensated op-amp with floating bulk and one extra inverter output stage.

In use, the comparator is loaded by the digital SA-block containing 10 small transistors $< 0.1pF$. The propagation time must be less than

$$\frac{1}{16ksamples/s \cdot 32cycles} \Rightarrow 1.95\mu s$$

The slew rate must with $V_{dd} = 3.3V$ be ten times faster

$0V \rightarrow 3.3V$ in $1.95\mu s \Rightarrow 1.68V/\mu \Rightarrow S_R = 16.8V/\mu s$

The input offset error must be $< 1/2$ LSB and the output swing should be 0.75 V from each rail. The gain must be at least $2^{10} = 1024$ times.

1. Current in the output stage is selected to meet slew rate requirements. This is done with the equation

$$I_2 = S_R \cdot C_L$$

resulting in $I_2 = 1.68\mu$ A.

2. $V_{bias} = V_{GS} - V_T$ is selected to 0.4 V (Equal to the buffer amplifier)

$$\left(\frac{W}{L}\right)_7 = \frac{I}{\frac{k_p}{2}(V_{GS} - V_T)^2}$$

$$\left(\frac{W}{L}\right)_7 = \frac{I_2}{\frac{k_p}{2}(0.4)^2} = 0.7$$

Output voltage swing is > 0.4 V from rail.

$$\left(\frac{W}{L}\right)_6 = \frac{I_2}{\frac{k_n}{2}(0.4)^2} = 0.3$$

The second-stage gain will be

$$A_2 = -\left(\frac{g_{m_6}}{g_{ds_6} + g_{ds_7}}\right)$$

$$= -\frac{\sqrt{2 \cdot k_n I_2 \left(\frac{W}{L}\right)_6}}{I_2(\lambda_p + \lambda_n)} = -50$$

The gain in the first stage must be 21.

Current I_1 for stage 1 is selected to be the same.

$$I_1 = I_2 = 0.5\mu A$$

Bias

$$V_{DS_3} = V_{GS_3} = V_{GS_4} = V_{DS_4} = V_{GS_6}$$

due to same current and same dimensions

$$\left(\frac{W}{L}\right)_3 = \left(\frac{W}{L}\right)_4 = \left(\frac{W}{L}\right)_6 \frac{I_1/2}{I_2} = 0.3 \cdot 0.5 = 0.15$$

$$\left(\frac{W}{L}\right)_5 = \frac{I_1}{\frac{k_p}{2}(0.4)^2} = 0.7$$

Input transistors M_1 and M_2 are determined from the total gain requirements. For safety the gain is multiplied with factor 7.

$$A_1 = \sqrt{\frac{2 \cdot k_p \cdot \left(\frac{W}{L}\right)_i}{I_{m_4}}} \cdot \left(\frac{1}{\gamma_1 + \gamma_4}\right)$$

$$\left(\frac{W}{L}\right)_i = ((\gamma_1 + \gamma_4)A_1)^2 \left(\frac{I_4}{2 \cdot kp}\right) =$$

$$((0.1) \cdot 21 \cdot 7)^2 \left(\frac{0.84 \cdot 10^{-6}}{2 \cdot 30 \cdot 10^{-6}}\right) = 3.025$$

The design was tested and tuned with simulation. If minimal dimension is selected to 2μ all transistor ratios can be found in table 6.

$\left(\frac{W}{L}\right)_i = 2.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_i = \frac{4}{2}$
$\left(\frac{W}{L}\right)_3 = \left(\frac{W}{L}\right)_4 = 0.29$	$\Rightarrow$	$\left(\frac{W}{L}\right)_{3,4} = \frac{2}{7}$
$\left(\frac{W}{L}\right)_6 = 2.5$	$\Rightarrow$	$\left(\frac{W}{L}\right)_6 = \frac{2}{0.8}$
$\left(\frac{W}{L}\right)_5 = 0.29$	$\Rightarrow$	$\left(\frac{W}{L}\right)_5 = \frac{2}{7}$
$\left(\frac{W}{L}\right)_7 = 1.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_7 = \frac{2.0}{2.0}$
$\left(\frac{W}{L}\right)_{10} = 2.5$	$\Rightarrow$	$\left(\frac{W}{L}\right)_{10} = \frac{2.0}{0.8}$
$\left(\frac{W}{L}\right)_{11} = 2.5$	$\Rightarrow$	$\left(\frac{W}{L}\right)_{11} = \frac{2.0}{0.8}$

Table 6: Transistor ratios

Simulation results:

$A_o = 601k \rightarrow 115.6dB$

Input offset $= -386\mu V$ (V_{in} 1.5 V DC)

Where an extra 50fP capacitance was added to the floating bulk.

C.4 Bootstrap bias generator

V_{bias} is defined by

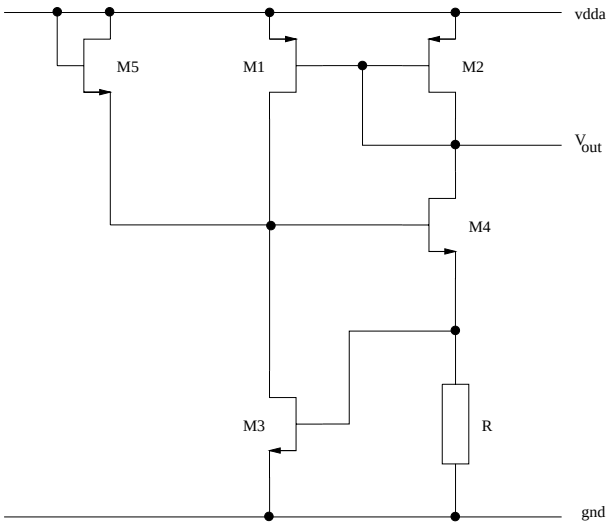


Figure 48: Schematic of vbias circuit

$$V_{bias} = V_{dda} - V_{out}$$

In our design V_{bias} is selected to $3.3 - 2.15 = 1.15V$. In the circuit, current i_1 (through M1) and i_2 (through M2) always have a constant relationship defined by the relative size of M1/M2. The current i_2, created by M2, is fed through the resistance R and the current i_1 from M1 must pass M3. The gate of M3 is connected to the resistance and therefore those two equations are connected to each other and the equilibrium point defined by the solution of

$$I_2 R = V_{T_3} + \left(\frac{2I_1}{k_n} \left(\frac{W}{L} \right)_3 \right)^{\frac{1}{2}}$$

$$I_Q = I_2 = \frac{V_{T_1}}{R}$$

This system have two solutions, one with I_1 and I_2 equal to zero and one with correct current in both stages. To secure right equilibrium point, M5 is added to the circuit and thereby the zero solution is eliminated during startup. The startup circuit is only active if the circuit is in the undesired equilibrium point. By inserting a small current through M3, mirrored with M1/M2 the equilibrium point is moved away from origo.

The two currents in the circuit are almost independent of V_{dda} and therefore the output V_{out} follow V_{dda} with a constant offset equal to V_{bias}.

If a large current is selected a small resistor is needed and a small current results in a large resistor. Low power trades large layout and in this application low power is preferred.

The current I_2 in resistor stage is selected to $10\mu A$ and the voltage over the resistor to $1.5V$.

Ratio for transistor M2 is determined by

$$I_D = \frac{1}{2}k_p\frac{W}{L}(V_{GS} - V_{T_p})^2(1 + \lambda V_{DS})$$

which gives a transistor ratio of

$$\left(\frac{W}{L}\right)_2 = \frac{2 \cdot 10\mu A}{30 \cdot 10^{-6}(V_{bias} - 0.75)^2(1 + 0.05 \cdot V_{bias})} = 4 = \frac{20}{5}$$

and the resistor

$$R = \frac{1.5}{10\mu A} = 150k\Omega$$

The mirrored current i_1 in the other stage is selected to be one tenth of i_2 resulting in the ratio

$$\left(\frac{W}{L}\right)_1 = 0.4 = \frac{5}{12.5}$$

Dimension of M3 is calculated by

$$I_D = \frac{1}{2}k_n\frac{W}{L}(V_{GS} - V_{T_n})^2(1 + \lambda V_{DS})$$

to

$$\left(\frac{W}{L}\right)_3 = \frac{2 \cdot 1\mu A}{70 \cdot 10^{-6}(1.50 - 0.88)^2(1 + 0.05 \cdot 3.3)} = 0.0638 = \frac{2}{31}$$

Size of M4 is not critical and therefore selected small.

$$\left(\frac{W}{L}\right)_4 = 2.0 = \frac{4}{2}$$

Transistor M5 controls the gate voltage of M4 and because I_D will be 0 the ratio is selected small.

$$\left(\frac{W}{L}\right)_5 = 2.0 = \frac{4}{2}$$

After some fine tuning in the simulator all ratios can be found in table 7. The circuit has been simulated a lot to ensure right startup and loading behavior. R is finally selected to $159K\Omega$.

$\left(\frac{W}{L}\right)_1 = 0.4$	$\Rightarrow$	$\left(\frac{W}{L}\right)_1 = \frac{5}{12.5}$
$\left(\frac{W}{L}\right)_2 = 4.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_2 = \frac{20}{5}$
$\left(\frac{W}{L}\right)_3 = 0.056$	$\Rightarrow$	$\left(\frac{W}{L}\right)_3 = \frac{2}{36}$
$\left(\frac{W}{L}\right)_4 = 2.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_4 = \frac{4}{2}$
$\left(\frac{W}{L}\right)_5 = 2.0$	$\Rightarrow$	$\left(\frac{W}{L}\right)_5 = \frac{2}{1}$

Table 7: Transistor ratios

Results from startup simulation with $R_L = 10M\Omega(0.2\mu A)$ after $5ms$.

$V_{out} = 2.143V$ $i_1 = 10.02\mu A$ $i_2 = 0.472\mu A$

A extra $1\mu A$ load decrease V_{out} with $18\mu V$.