# Chapter 3

# METHODS

## 3.1 CHAPTER OBJECTIVES

This chapter discusses the methods that have been used to develop the elephant recording collar as well as the automatic elephant rumble detection and pitch tracking algorithm. The details of the electronic design of the collar will be discussed in Section 3.3. The mechanical design of the collar will be examined in Section 3.4 while the development of a rumble detection and pitch tracking algorithm follows in Section 3.5.

## 3.2 INTRODUCTION

One of the primary aims of this study was the development of a instrument that can record high volumes of continuous acoustic data in unfavourable circumstances. It has been decided that the recording device will be mounted within an elephant collar. The aim behind the development of the collar is to assist zoologists in obtaining large quantities of acoustic data from wild elephants that can be used in further research.

Before a design was made, thought had to be given to the specific mechanical and functional requirements that such a collar would have to satisfy. The requirements were identified by communication with elephant researchers and from scientific literature.

The electronic and mechanical designs were made with these requirements in mind.

The development of the automatic rumble and pitch tracking algorithm was done by first gaining insight into the nature of elephant rumbles by studying elephant vocalization recordings and then using the newly gained knowledge to devise a rumble detection algorithm based on speech detection techniques.

## 3.3 ELECTRONIC DESIGN

The requirements of the electronic design will be discussed first (3.3.1). Important specifications were determined from this section and were used to devise a concept design for the electronic system (3.3.2). A detailed discussion is given on the design considerations and choices that had to be made when implementing each of the elements shown in the concept design into a realistic electronic design (3.3.3). A functional description is given for each electronic module with a brief discussion of standard electronic design theory where applicable. The way in which the chosen components satisfy the desirable design characteristics of each module is discussed for every module. The operation of the CompactFlash (CF) memory cards is discussed in more detail as these devices are central to the design and has a standard interface that masks the operation of the internal flash technology.

The functional considerations regaring the printed circuit board are discussed (3.3.4). A functional description of the microcontroller software are given (3.3.5). A detailed discussion on the operation of the FAT32 file system is given since it plays a central role in the software design.

### 3.3.1 Design requirements

A collar that can record high volumes of continuous elephant acoustic data in unfavourable circumstances needed to be developed. The implication on the electronics of such an elephant collar will now be considered.

Firstly it is important to remember that a wild elephant would have to be tranquillized when the collar is fitted and again when the collar is removed. This is an expensive exercise that needs clearance from an ethics committee. Once a recording collar is fitted, it would be desirable that it keeps on recording continuously for a long enough period to make the experiment worthwhile. A minimum recording period of 90 days should ensure that enough data can be gathered.

The device should record the bandwidth of sound that an elephant produces. Elephant rumbles have frequency components as low 8 Hz while their trumpet sound has frequencies of up to 1.150 kHz. All the frequencies between 8 Hz and 1.150 kHz should be recorded. The sampling rate should, however, be kept as low as possible to ensure that memory space is not wasted. A sampling frequency of 3 kHz should be in order, giving an anti-aliasing filter enough spectral space to suppress any frequencies above the Nyquist frequency (1.5 kHz).

Sound should be recorded at a good enough resolution to ensure that its quality is acceptable for further processing. On the other hand, recording at a higher than required resolution would waste memory space. A resolution of 16 bits provides high quality recordings for human speech, so it would be a good choice for the recording of elephant sounds.

Enough memory storage should be provided within the collar to ensure that the collar can keep recording for the required period of time. A minimum capacity of 43 GB is required to record the sound. (This was calculated using Equation 3.7 given in Section 3.3.3.5). It should also be remembered that the collar should not cause discomfort to the elephant wearing it. This implies that the weight and size of the collar should be restricted.

If a recording device is going to be fitted on an elephant, the possibilities of recording extra information could be explored. Any information regarding the circumstances in which a sound recording is made might prove valuable. A GPS and a thermometer could be incorporated into the device to record the outside temperature and the location of the elephant.

Table 3.1 gives a summary of the most important requirements of the collar electronics. These requirements were considered before a specific electronic configuration were chosen.

Table 3.1: Specifications derived for the recording collar.

| **Parameter** | **Consideration** | **Specifications** |
|---|---|---|
| Recording duration | The elephant needs to be tranquilized before fitting or taking off the collar | 90 days (or more) |
| Sound bandwidth | Infrasonic rumbles as well as trumpet vocalizations should be recorded | 8 – 1150 Hz (3000 Hz sampling) |
| Sound resolution | The recorded sound will need to be clearly defined in order to be processed | 16 bits |
| Memory storage | There should be enough memory to last for the duration of the experiment and it should be physically robust | 42 Gigabytes (or more) shock resistant memory |
| Power dissipation | The battery should last for the duration of the experiment | 33 mA (or less) |
| Weight | The collar should not hinder nor injure the elephant | 10 kg (or less) |
| Size | The collar should not obstruct the elephant's movements | 25x25x30 cm (or smaller) |
| Added features | The more information that accompanies the sound files the better | GPS, and digital thermometer |

### 3.3.2 Concept design

With the specifications shown in Table 3.1 in mind a concept design of the electronic system were done. The concept design is shown in Figure 3.1. A high level functional description of the concept design will now be given. The details of each of the elements will be given under the corresponding heading.

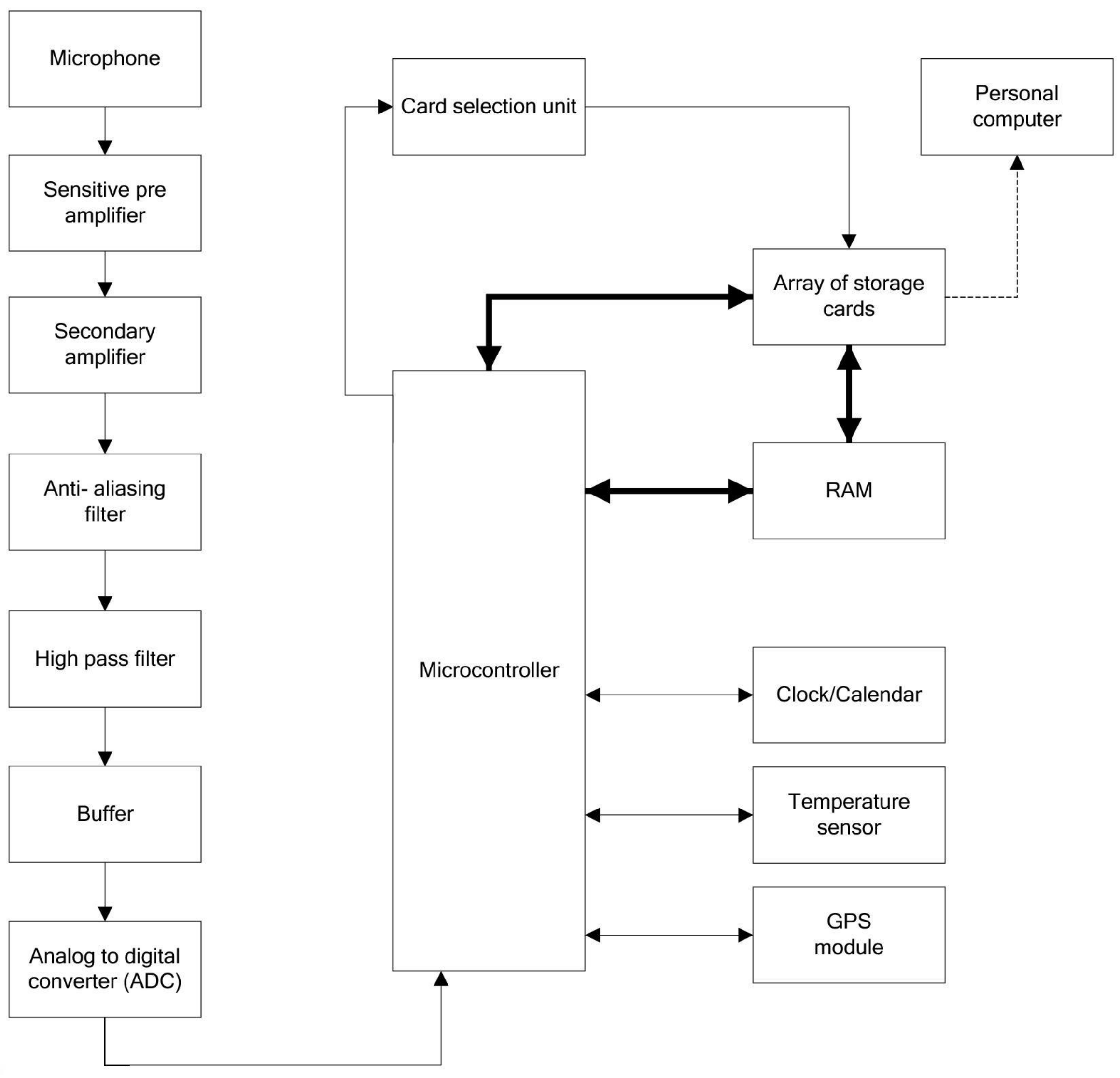


Figure 3.1: Hardware concept design block diagram.

A microphone converts sound waves from the surrounding area into an analogue electrical signal. The signal coming from the microphone needs to be amplified to a voltage signal with useful amplitude. This is done by using a sensitive preamplifier to effectively amplify the small signal directly at the output of the microphone. Another variable amplifier is then used to tune the gain to the desired value.

A sixth order Butterworth low pass filter was used as an anti-aliasing filter to filter out all frequency components higher than half of the Nyquist frequency. A high pass filter and a buffer were used to centre the filtered signal around 0 V for the bipolar ADC (Analogue to Digital Converter) and to provide a low input impedance to the analogue input of the ADC respectively. The ADC converts the analogue sound signal to a sixteen bit digital word. The timer module in the microcontroller generates a conversion pulse at a frequency of 3 kHz to the ADC to regulate its frequency of conversion. The microcontroller controls the flow of data from the ADC to the storage media and controls the timing and interaction of all other digital components in the system.

An array of solid state memory cards are used for data storage. The microcontroller controls the hardware responsible for selecting the memory cards one at a time so that data may be written to a single card until its capacity has been reached and then move on to the next card. A digital thermometer is used to measure the external temperature and a GPS module determines the positional coordinates as well as time and date information.

### 3.3.3 Detailed discussion

In order to realize the hardware concept design, every block of the block diagram in Figure 3.1 were designed and implemented in hardware. The hardware was designed to use as little power as possible and the idea was to design an appropriate power supply (set of batteries) only after the power consumption of the final design had been tested.

#### 3.3.3.1 Microphone

The microphone that was used for this system should have the following characteristics:

1. A dynamic range of at least 16 Hz to 1.15 kHz (the frequency range of elephant vocalizations).
2. Rugged microphones should be chosen that has good resistance to physical impacts.
3. The microphone housing should be water and mud resistant.

A microphone manufactured by Knowles (model MR-23793) was chosen. The microphone was specified by the manufacturer to have good resistance to physical shocks, be waterproof for up to 15 metres and to resist the effects of mud, sand and salt encrustation. It also had a flat frequency response throughout the applicable bandwidth which made it a good choice for use in the elephant collar.

A circuit was designed to generate the microphone's biasing voltage and to centre the output signal of the microphone at the ground potential. A simple voltage divider circuit was used to provide the biasing voltage for the microphone. Equation 3.1 was used to determine the values of the resistors to be used for the voltage divider.

$$V_M = V_{CC}\left(\frac{R_2}{R_1 + R_2}\right). \tag{3.1}$$

A simple RC high-pass filter with a very low 3 dB cut-off frequency of 1 Hz was used to remove the DC voltage component from the output of the microphone. Equation 3.2 was used to determine the value of the resistor for the high-pass filter. In the equation, $f_c$ is the 3 dB cut-off frequency, $R$ is the parallel resistor and $C$ is the series capacitor of the high-pass filter.

$$f_c = \frac{1}{2\pi RC}. \tag{3.2}$$

A cut-off frequency of 1 Hz was chosen to ensure that none of the spectral information from the elephant vocalizations would get lost. A capacitor value of 100 nF was arbitrarily chosen which gives a resistor value of approximately 159 kΩ. As can be seen from the circuit diagram shown in Figure 3.2, the voltage divider is buffered by an operational amplifier (more information on the choice of a specific operation amplifier will be given with the amplifier discussion). This is done so that the microphone may see a low input impedance on the biasing pin and will thus be able to draw current without effecting the voltage divider network. The capacitor between the biasing output at the voltage divider and ground will filter out any ripple that may be present on the signal.

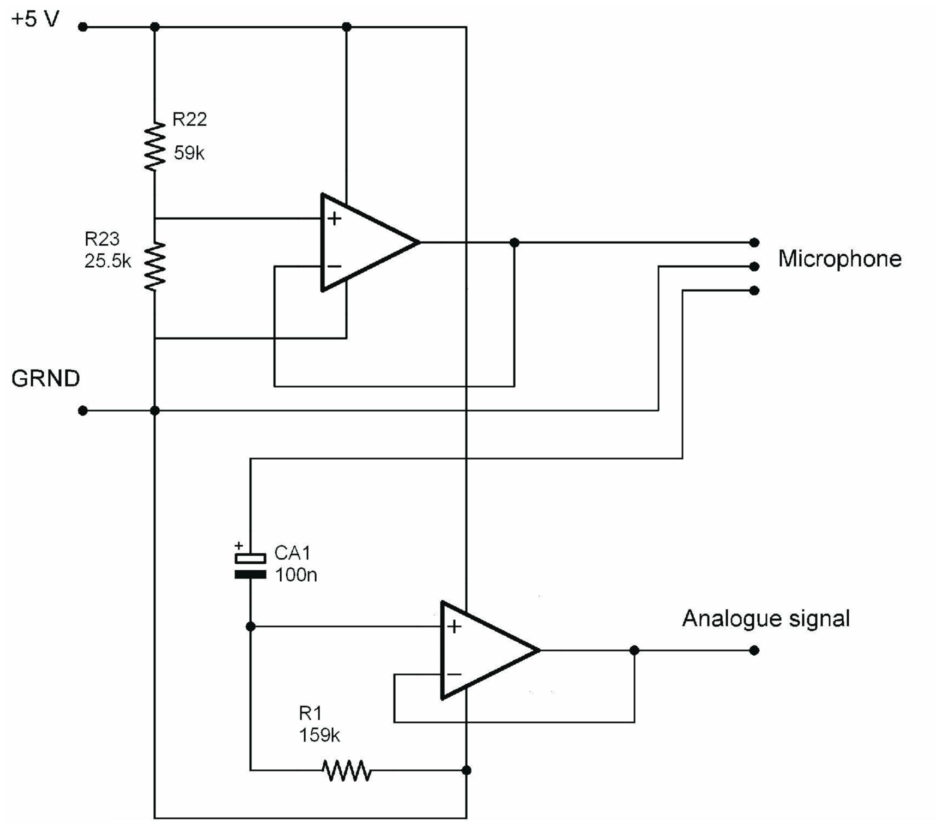


Figure 3.2: A schematic of the microphone conditioning circuit.

#### 3.3.3.2 Amplifier

The analogue signal received from the microphone is in the tens of millivolts range, which requires that a sensitive amplifier was designed to amplify the signal to a useful magnitude.

An amplifier with AGC (Automatic Gain Control) was considered. AGC has the advantage of being able to record both soft sounds and loud sounds by automatically adjusting the gain level of the amplifier. However, the sound intensity is an important characteristic of the elephant vocalizations (e.g. this may be used to estimate the distance of a different elephant). This means that if an AGC is used, the gain level should be continuously recorded with the sound data to ensure that the sound intensity at a given instant can be established. Therefore, the limited storage capacity onboard the recording collar prohibits the use of AGC.

The second option is to set the amplifier to an optimal sensitivity level for the vocalizations of the particular elephant wearing the collar. This option has been chosen, because the microphone will always be at a relatively constant distance from the source of the sound.

A design choice was made to use an operational amplifier for the amplification of the signal from the microphone. This choice was made because of the ease of implementation and the ideal input and output impedances that an operational amplifier provides. A general purpose operational amplifier has a drain current of up to 5 mA. The power consumption of the amplifier that is used in this application has to be as low as possible to conserve battery power. MC33171 micropower operational amplifiers were selected for the design. These operational amplifiers are specified to have a drain current of less than 100 $\mu$A which is at present the operational amplifier with the lowest power consumption of all those available that can operate at the desired voltage.

Two inverting amplifiers were cascaded with the first one being a very sensitive amplifier and the second one adding gain to the total amplifying circuit. Equation 3.3 was used to determine the gain of each amplifier with $G$ being the gain of the amplifier, $R_1$ the resistor connected to the input signal and $R_2$ the feedback resistor.

$$G = \frac{R_2}{R_1}. \tag{3.3}$$

Each amplifier was designed to have a gain adjustable from 0 to -20 V/V by choosing $R_1$ 10 k$\Omega$ and using a 200 k$\Omega$ potentiometer for $R_2$. This results in a total achievable gain of 400 V/V. The resistors connecting the non-inverting inputs of the operational amplifiers to ground were determined by Equation 3.4.

$$R_x = R_1 || R_2. \tag{3.4}$$

This resistor value optimizes the offset voltage of the amplifier. Since the value of the feedback resistor will almost certainly always be much larger than the input resistor, a value close the input resistor's value of 10 kΩ has been chosen for $R_x$. A value off 10 kΩ has been chosen for $R_x$ in both amplifiers.

#### 3.3.3.3 Anti-aliasing filter

The anti-aliasing filter is a low pass filter that should ensure that no spectral overlap occurs in the recorded signal. All frequencies with a value higher than half of the sampling frequency should be suppressed to avoid aliasing when the signal is sampled. The sampling rate is 3 kHz, thus the highest frequency in the sampled signal is limited to 1.5 kHz. It will be desirable to suppress frequencies higher than 1.5 kHz with at least 20 dB. The highest frequency that needs to be recorded is 1 kHz. This means that the filter can start to cut off at 1 kHz and should reach 20 dB attenuation within 500 Hz.

The first design choice that had to be made was to use either a digital or an analogue filter. A digital filter provides an excellent anti-aliasing filter and needs only a small resistor-capacitor filter after its output. Such a filter can be found inside an audio codec chip that also houses an ADC and gives a digitized sound signal as output. It does however need a high frequency clock source to drive it and the ones that are available drains 15 mA of current which makes its power consumption too high (remembering that the total current drain of the system should remain less than 33 mA).

Both Butterworth and Chebychev analogue filters were considered as alternatives. A Butterworth filter has the advantage of having a maximally flat pass band, but its cut-off rate is slower than that of an equal order Chebychev filter. A Chebychev filter has a much steeper cut off-rate but has ripple in the pass band.

A sixth order Butterworth filter was chosen. The flatness of its pass band and the ease of implementation were the main reasons for the choice. Even small tolerances in values of components may result in a higher order Chebychev filter becoming unstable. In the initial realization, three almost identical second order sections were implemented. Table 3.2 shows the frequency and gain values for each of these second order sections needed to realize a sixth order Butterworth filter with a 3dB cut off at 1 kHz.

Table 3.2: Cut-off frequency and gain of each second order section of the filter.

| **Section no** | $f_c$ **(Cut off frequency)** | $G$ **(Gain)** |
|---|---|---|
| 1 | 1 kHz | 1.068 |
| 2 | 1 kHz | 1.586 |
| 3 | 1 kHz | 2.483 |

Equation 3.5 is used to calculate the frequency dependent component values while Equation 3.6 calculates the gain of the section.

$$f_c = \frac{1}{2\pi RC}. \tag{3.5}$$

$$G = \frac{R_2}{R_1}. \tag{3.6}$$

Figure 3.3 shows one second order section of the Butterworth filter. Three of these sections, each with a different gain (as shown in Table 3.2), were connected in series to realize the Butterworth filter.

#### 3.3.3.4 Analogue to digital converter

The function of the ADC is to convert the amplified analogue signal received from the microphone into a digital signal. A wide variety of ADC's are available in IC (integrated circuit) packages. The main difference between the available ADC's are the data output method, input range, speed of conversion, and resolution and power consumption.

The speed of conversion needs to be fast enough so that a complete conversion can be done and the digital data can be read and stored during a single sampling period. Since the sampling frequency is 3 kHz, the rate of conversion should be greater than 3000 conversions per second. The resolution of the ADC determines its SNR (Signal-to-Noise Ratio). A 16 bit ADC should be used, which gives a SNR of 98 dB.

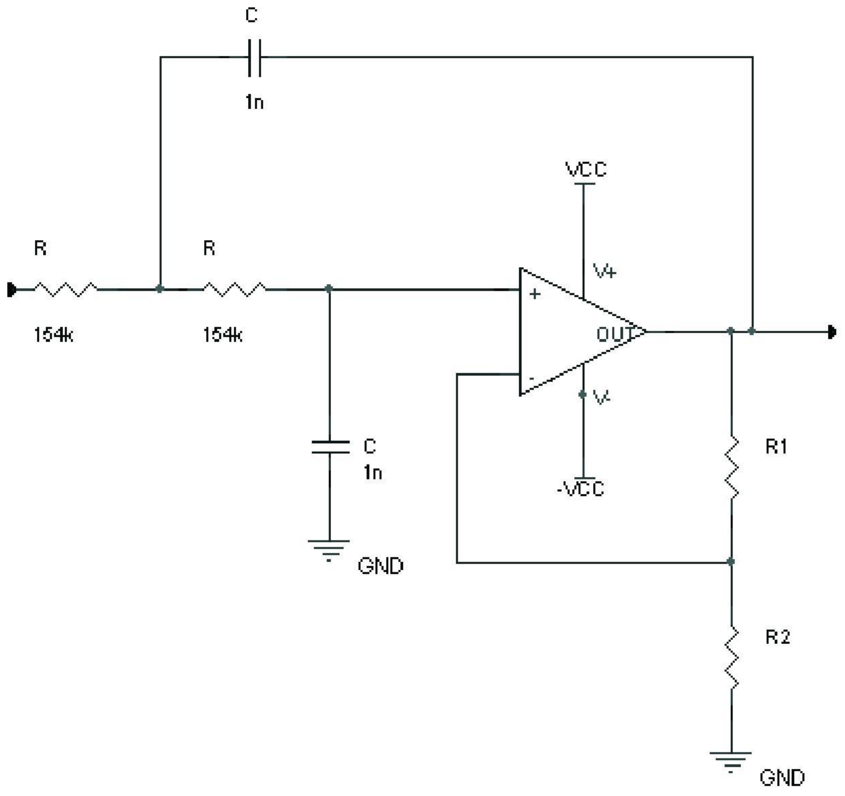


Figure 3.3: A single second order section of the Butterworth filter.

#### 3.3.3.5 Data storage

The digital data received from the ADC have to be stored on permanent storage media. The permanent storage media should have the following characteristics:

1. A storage capacity of more than 43 GB is needed to store data continuously for three months.
2. Resistance against physical impacts is a necessity.
3. The storage device should be as power efficient as possible.
4. Memory with small physical size is required.

The sampling frequency is 3 kHz and 2 bytes of data will be stored 3000 times per second. Using Equation 3.7, this results in approximately 43 GB of memory required for continuously recording over a 90 day period.

$$\text{Capacity} = Bytes/s \times 3000Hz \times 60s \times 60min \times 24h \times days. \tag{3.7}$$

A notebook hard drive was considered as one alternative. It is relatively small in size and will have ample storage capacity. However, because a hard drive has moving parts it could be the weakest link when considering the robustness of the system. It is unclear whether a hard drive would be able to survive the physical impacts that the collar would be exposed to while on an elephant. The power usage of a hard drive is also very high in comparison to some solid-state memory cards.

Different solid-state memory options were available. RamSan is super fast RAM (Random Access Memory) memory that has reached capacities of more than 1 Terabytes. Even though it is solid state memory and has more than enough storage space, the size and high power usage of these devices makes it unsuitable for this application. Flash memory seems like the best choice for the application at hand, since it is mechanically robust, small in size and uses little power. The memory capacity on a singe flash memory device will not be sufficient. It has therefore been decided that an array of flash memory devices would be used to provide enough storage memory to the system.

Universal Serial Bus (USB) flash drives, SD memory cards and CF memory cards were but a few of the other available options.

After some consideration, CF cards have been chosen as the storage media that would be most suitable for the recording system. Mini hard drives with a CF interface are also available, so using CF cards would keep the option open for experimenting with (less expensive) hard drives to see if they will survive for the duration of the recording period. An array of four CF cards will be used to ensure that required amount of memory space is provided.

The CompactFlash card specification was formulated by the CompactFlash Association in 1995. CF cards are compact, removable non-volatile memory cards. It has 50 interface pins and is roughly the size of a matchbook. Internally, the CF card consists of an interfacing microcontroller, a buffer and varying amounts of non-volatile memory. The memory capacity of CF cards is ever increasing and 32 GB CF cards are currently available. The fact that CF cards have the highest storage capacity of all flash memory cards and this together with its low power consumption is the main reason why it has been chosen for this design.

The internal microcontroller of the CF card does the work of storing and accessing the data onto and from the physical memory according to commands given to it through the interface pins. This makes it possible to access CF cards with different memory capacities and technologies with the same instructions so that future CF cards that use new technology can still be accessed in the same way as before. The CF card's internal microcontroller also makes it possible to access the card in a number of different modes, including Common Memory mode and true IDE mode. When the card operates in true IDE mode it can be connected to an IDE bus and a PC will see it as a hard drive. When configured to operate in the Common Memory mode (as is the case for the elephant collar), the CF card operates with an 8 bit data bus instead of the 16 bit data bus when used in IDE mode. This enables an 8-bit microcontroller to access the card.

The CF card can be completely controlled through eight control registers by loading them with appropriate data. Table 3.3 shows the address and name of each of the eight addressable registers. As can be seen from the table, only three address lines (A0 to A2) are needed to access the control registers and thus control the whole CF card.

Table 3.3: The eight control registers of a CF card and their three-bit addresses.

| -REG | A10 | A9-A4 | A3 | A2 | A1 | A0 | Offset | -OE=0 | -WE=0 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | X | 0 | 0 | 0 | 0 | 0 | Even RD Data | Even WR Data |
| 1 | 0 | X | 0 | 0 | 0 | 1 | 1 | Error | Features |
| 1 | 0 | X | 0 | 0 | 1 | 0 | 2 | Sector Count | Sector Count |
| 1 | 0 | X | 0 | 0 | 1 | 1 | 3 | Sector No. | Sector No. |
| 1 | 0 | X | 0 | 1 | 0 | 0 | 4 | Cylinder Low | Cylinder Low |
| 1 | 0 | X | 0 | 1 | 0 | 1 | 5 | Cylinder High | Cylinder High |
| 1 | 0 | X | 0 | 1 | 1 | 0 | 6 | Select Card/Head | Select Card/Head |
| 1 | 0 | X | 0 | 1 | 1 | 1 | 7 | Status | Command |

To load these registers with data, the correct address is placed on the 3-bit address line and a read strobe is given to read the data into the register. After the first seven bytes have been loaded with sector address information, the Command register is loaded with a certain command (usually either a Read Buffer or Write Buffer command). Address zero (the Data register address) is then selected by the three address lines to enable a card buffer to be read or written to. If a Read command is written to the Command register, the CF card's memory buffer is loaded with the contents of the sector to which the address registers is pointing and the information contained in that sector can be read byte by byte by giving a Read pulse until the desired information is reached. If a Write command is written to the Command register, an empty 512 bit buffer is presented and one byte of data is read into the buffer every time the Write pulse is given. When the buffer is completely filled with written data, the internal microcontroller of the CF card automatically moves the written data from the buffer to the non-volatile memory of the card. The next sector address then needs to be loaded into the address registers and another command written into the Command register.

Although the register names imply the SCH (Sector Cylinder Head) sector addressing method, LBA (Logical Block Address) addressing can be selected by setting the three most significant bits of the Head register. LBA is preferred to SCH because with LBA each sector of the disk gets its own specific number starting at zero and counting up the card capacity. With a 27 bit LBA, more than 64 GB of memory may be addressed. Using LBA addressing, the Sector No. register becomes the LBA first byte register, the Cylinder Low register becomes the LBA second byte register and the Cylinder high becomes the LBA third byte register.

When in Common Memory mode, the CF card can operate using only 17 of its 50 pins: the eight data pins, three address pins, read, write, ready/busy, reset, card detect and card select pins. Address lines 3 to 10 are connected to ground and the second cards select pin and the Reg pin are connected to Vcc. All the other pins are left unconnected.

The ready/busy pin is tested to determine whether the card is ready to accept new data. Read and write pin are used to give either read or write pulses. The reset pin can be pulsed to reset the internal microcontroller of the CF card. The card detection pin is internally routed to ground. It can be connected to Vcc through a pull up resistor and the value at the pin can be read to determine if a card is present. If a card is present, the resistor will be grounded and a low signal will be read on the pin. If a card is not inserted, the resistor has one end connected to Vcc and the other open end is read as a high signal, which indicates to the system's controller that no card is present. The card select pin is used to activate a CF card and connect the data bus to the card. If the card is not enabled, the data bus is disconnected and in a state of high impedance. This pin plays a very important role in the designed system since all the CF cards and the RAM shares the same 8-bit data bus.

A problem that had to be addressed was the fact that a single CF card would not have enough storage capacity to store all the data for the desired period of at least 90 days. One CF card has a maximum storage capacity of 16 GB while around 40 GB of storage space is needed. Connecting multiple CF cards onto a shared 8-bit data bus can solve this problem by selecting only one card at a time using the card select pin. When operating in this way all the pins of each card can be connected to each other accept for the card select pin, the card detect pin and the ready/busy pin which cannot be shared. When using 4 CF cards, a total memory capacity of 128 GB can be achieved.

#### 3.3.3.6 Card detection and selection logic

As mentioned in the previous section, each CF card needs a dedicated card select, card detect and ready/busy pin. This results in an extra 12 I/O pins needed on the microcontroller. To save I/O pins on the microcontroller, a card selection module was developed. The card enable pins are input pins that need to be written to and the ready/busy and card detect pins are output type pins that need to be read. The card enable pin on a CF card allows a single card in an array of cards to be selected for reading and writing operations. If a low signal is applied to the active low card enable pin, the card is active and ready to send or receive data. If on the other hand a high signal is applied to the card enable pin the card disconnects its data and address busses and goes into an inactive low power state.

All the card enable pins were connected to the first section of a 74139 2 to 4 line decoder with active low outputs. Two address lines from the microprocessor control the 4 output pins. The output pin to which the applied address corresponds to goes low while all the other pins remain high. In this way each of the 4 CF cards can be selected using only two I/O pins from the microcontroller. If the chip enable input of the 74HCT139 is deactivated, all its output pins goes to the high state.

Now that each card can be selected as desired, the dedicated output signals from each card needs to be linked with the microprocessor using the minimum amount of I/O pins. This has been done using a 74153 dual 4-bit multiplexer. When a card is selected, it's ready/busy and card select pin is automatically routed to the respective I/O pins of the microcontroller and the microcontroller can function in the same way as it would if only a single card were present.

The card detect pin of a CF card is internally routed to ground. When a card is inserted the pin is routed to ground and the voltage on the pin changes from high to low so that it may be detected that a card is present in that slot. A high resistor value of 159 k$\Omega$ was chosen to put in series with the card detection pin to ensure minimum current drain. Using Ohm's law, a current of only 32 uA is drained through each resistor when a card is present and has virtually no voltage drop over it when a card is not present.

#### 3.3.3.7 RAM memory

The fact that a CF card drains 30 mA or more when in writing mode, means that the system must be designed so that the time that the card spends in writing mode is minimized.

It had been decided to add 512 kB of static RAM to the system. This enables the system to store 1024 sectors of data in the external RAM. While the digital sound samples are stored in the RAM the CF card may be disabled, putting it in a low power state. When the RAM is filled to capacity the data will be transferred to the CF card at maximal speed. The on time of the CF card will only be a small fraction of the time it takes to accumulate the 512 kB of data samples. CF cards uses 200 mA of current when in write or read mode, so using the low power RAM (100 uA) reduces power consumption considerably.

The 62512 512kB RAM IC was used to supply the external RAM. The RAM chip is very power efficient, since it drains only 100 $\mu$A. To save I/O pins on the microcontroller, two 8-bit latches (74HCT377's ) and a 4-bit latch (74HCT75) were connected to the common 8-bit data bus shared by the CF cards. The data pins of the RAM chip were also connected to the common data bus. To address the RAM, all the devices on the common data bus was disabled, except the three latches. The data bus was used to clock and hold the first part of the RAM address on the first latch and then clock and hold the second part of the address on the second latch and then on the third latch. Now that an address for the desired bit position on the RAM had been applied, the RAM unit is enabled. To write data into the RAM the data is presented on the data bus and stored to the RAM by pulsing its write pin. To read data out of the RAM, the read pin of the RAM is pulsed to put the data from the RAM onto the data bus.

#### 3.3.3.8 Microprocessor

It was considered to use a single board computer to manage the storage of the data on the CF cards. This option would make the task of storing sound files in on a file system much easier. However, the high power usage and large physical size of a single board computer together with high cost made it an unfeasible option.

Therefore, it was decided that a microprocessor would manage the flow of data from the ADC to the storage media. It is important that the microprocessor consumes as little power as possible and that it has enough I/O pins. An 8-bit data/address bus, a 3-bit instruction bus and at least 6 other I/O pins are needed to establish communication between the microprocessor and the CF card. Another three pins are needed for the communication with the ADC. For accessing multiple CF cards, multiplexers and demultiplexers would be needed which also requires 4 pins. This amounts to 24 I/O pins, but since it is desirable to have a modular design in which other peripherals could later be added to the system if the need arises, it would be wise to have at least 30 I/O pins available. Using the CF card in its common memory mode allows us to use an 8-bit microcontroller, which is less power consuming than 16-bit or 32 bit processors. The PIC16f877 8-bit microcontroller was chosen because of its low power consumption. It has 33 I/O pins, which will allow extra peripherals to be added to the system. The PIC was programmed in assembly language using Microchip's MPLAB software and using the MPASMWIN assembler. The software design and implementation on the microcontroller will be discussed in the Section 3.3.5.

#### 3.3.3.9 Thermometer

The advantages of the addition of a digital thermometer to the system has been discussed in Section 3.3.1. A device with low power consumption, an accuracy and resolution of at least 1 °C in the correct temperature range should be used.

Maxim's Ds18B20 low power digital thermometer was chosen. It communicates with the microcontroller using a one-wire interface, so that only one I/O pin of the microcontroller is used to send and receive data. The device can be configured to operate in a parasitic power mode where the device does not need to be connected to an external power source but charges its internal battery each time a high data pulse is received. The device has a resolution of 0.0625 °C and an accuracy of 0.5 °C. The device can measure temperatures ranging from -50 °C up to 125 °C.

#### 3.3.3.10 GPS module

The U-blox LEA-LA module was chosen used because of its small size and the fact that it had the lowest power consumption of all available modules. The LEA-LA is equipped with two Universal Serial Asynchronous Receive Transmit (USART) ports. The PIC16F877 is also equipped with an USART port and communication is established using the RX and TX pins of both devices.

An U-blox customized GPS protocol called UBX is used for communication with the microcontroller. An UBX command can be sent to request positional coordinates, as well as time and date information. When the GPS is in normal operating mode, it draws a current of approximately 150 mA. It should thus only be used to acquire the necessary information and then be switched into sleeping mode where less than 1mA of current is drawn.

### 3.3.4 Printed circuit board design

All the main electronics are situated on a main system board while the CF card connectors and the card selection and detection modules are housed on a separate connector board. These two boards both had measurements of 140x180 mm. They were mounted on top of one another to ensure that the size of the collar remained within the desired specifications.

Both printed circuit boards (PCBs) were designed in OrCad 10 using four conducting layers. The top and bottom layers are used as routing layers, while the two layers in the middle are used as a ground layer and a power layer. The four-layered approach was chosen for two reasons. First, having more layers allows components to be placed closer together allowing for a smaller PCB to be manufactured. Second, having a ground and power layer increases the stability of operation of the electronics. Variations on the power lines are counteracted by the capacitive effect of the two power layers. The routing on the top layer is horizontal while the routing on the bottom layer is vertical to ensure minimal signal interference.

### 3.3.5 Microprocessor software design

All the digital hardware described in the previous section are controlled by the microcontroller. It has the task of coordinating all the different components of the system and to control the flow of data from the microphone to the CF cards. All the software was written for the PIC16F877 microcontroller using assembly language. All programming was done in Microchip's MPLAB.

Even though the software was written specifically for the PIC16F877 microcontroller, only the functional design of the software will be discussed. The same principles that have been used in the software design for this project could later be used even if a different microprocessor was to be selected.

In principle, the software should be designed to regulate the following operations:

1. Data protection
2. File system management
3. File structure control
4. Data flow control

Data protection should be implemented to ensure that previously recorded data can not be overwritten by accident. File system management ensures that the data that is stored on the memory cards will be accessible from any Windows operated computer. It also ensures that the data can be transferred from the cards to a computer's hard drive. In this case the FAT32 file system was implemented. The file structure is used within the file system and provides a way to store the sound data in a standardized manner so that it can be played or processed correctly by other computer programs.

The software controls the operation of the different hardware modules and also determines in what format the data is stored on the memory cards. The following subsections describes each software module in more detail.

#### 3.3.5.1 Protecting the recorded data

As soon as the power to the system is switched on, some microcontroller specific tasks are performed to set it up for correct operation.

After the initialization, the system waits for a record command. If the system simply started to record data as soon as the power has been switched on, previously recorded data that had not yet been downloaded would be overwritten. Also, the power to the system might be switched on in advance, but it will not be desirable to start recording right away. For these reasons recording should only start after the user of the device has given a record command.

Data protection have been done by forcing the user to first press a push button for a certain duration of time before the recording process is started. The flow diagram of Figure 3.4 shows how the user will enable the start of the recording period. After the power has been switched on, a dedicated pin will be configured as an input pin and is pulled low by an external resistor. This pin is connected to the base of a transistor that can drive a LED. When the pushbutton is depressed, the signal on the LED pin will change to the high state. The program will go into a loop and poll the state of the signal on the LED pin continuously to determine whether the button has been depressed. When the button is pressed the change of state on the pin will be detected and the program will set a register with a value of 8 to be used as a counter. It will then go into a subroutine to determine whether the button is pressed for longer than 4 seconds, by calling a delay of a half second and then testing (repeat eight times) whether the button is still being pressed. If the button is still pressed after half a second, the counter is decremented and another half-second delay is called. This process is repeated until the counter reaches zero or if the button is released before the 4 seconds have elapsed. If after any one of the half-second delays it is detected that the button is not pressed anymore the program will return to the loop and wait for the next time the button is pressed where after the whole process will be repeated again. The reason why the button has to be pressed for 4 seconds is to ensure that the device doesn't start recording if the button have accidentally been touched. If the counter reached zero and the button is still being depressed, a high signal will be placed on the pin that will drive the LED.

The shining LED will alert the operator that the device has started recording. The LED is also connected through the switch in order not to waste power, so after the operator has released the button the LED will be deactivated. The recording of data will now commence and the push button will have no further influence until the power has been switched off and on again.

#### 3.3.5.2 Memory card detection and selection

As has been mentioned in the hardware design section of this report, there are eight CF cards connected on a single data bus. The address bus and the read and write pins are also shared by all the cards. Each card, however, needs its own dedicated card detect, card enable and ready/busy pins. This is achieved by using a 3 to 8 bit

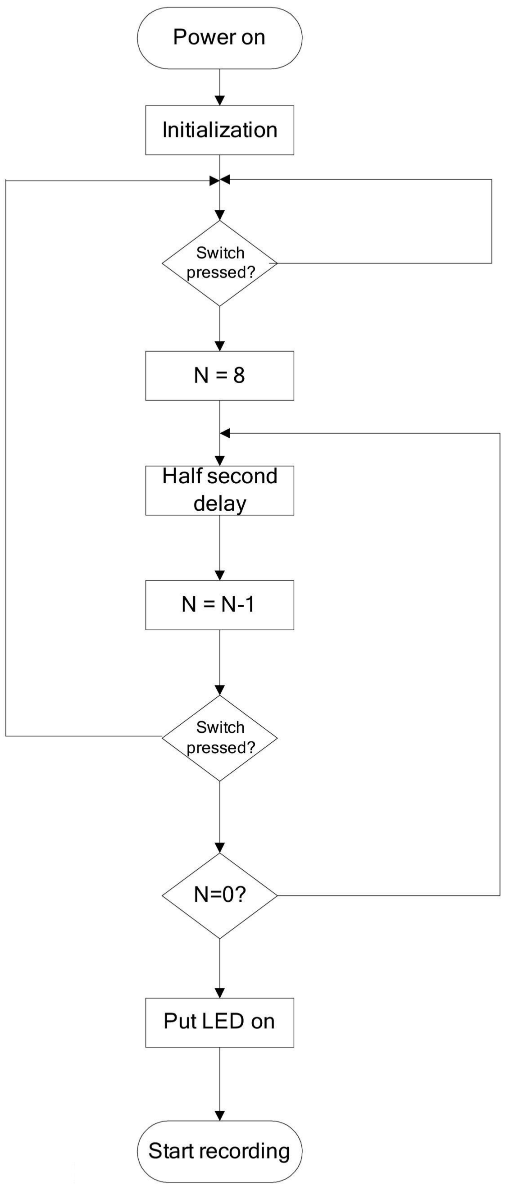


Figure 3.4: A flow diagram describing the program directly after the power to the system has been switched on.

line decoder and two 8-bit multiplexers. Figure 3.5 shows a flow diagram of how the software selects and detects CF cards.

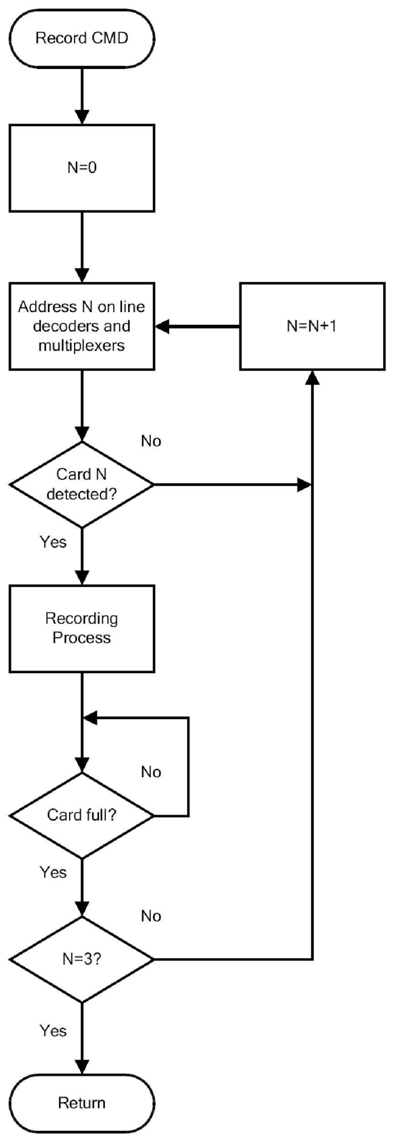


Figure 3.5: A flow diagram describing how specific CF cards are detected and selected for use.

When the record command is given, the first CF card socket is selected by writing zeros to all three address lines of the 3 to 8 decoder. This results in a low signal being sent the card enable pin of the first CF socket and high signals being sent to the other CF socket's card enable pins. The same addresses are sent to the multiplexers to connect the correct socket's ready/busy pin and card detect pin to the microcontroller. After the first CF socket has been selected, the card detect pin is used to determine whether a card is present in that specific socket. If a card is not detected the address of the line decoder is increased and the next CF socket is selected. The card detect pin of this socket is used to detect whether a CF card is present and this process repeats itself until a card is detected or until all eight the CF sockets have been checked. As soon as a card is detected, the program jumps out of the loop and continues with the rest of the prerecording routines. Data will be recorded onto the card until its capacity is reached. The program will then jump back to the card detection loop to find the next socket that contains a CF card. This process will be repeated until the eighth CF socket has been checked.

#### 3.3.5.3 CF card read and write operations

Arguably the most important task of the microprocessor is to write data to the CF cards. As was discussed in the theory section, the entire CF card may be accessed using only eight control registers. The address of the sector that needs to be accessed is loaded into four address registers and either a read or write instruction is loaded into the command register. The data register is then selected and all data transfers take place through this register. Data is clocked into the CF card using a read or write pulse. The process is described by the flow diagram of Figure 3.6. The read and write routines were written based on a piece code published on the Internet by Mark Samuels.

After the LBA address of the desired sector has been entered, either a read or write command is loaded into the command register. At this point, if a read command was given, the CF card will load the addressed sector in to its data buffer. The data may then be read out sequentially until the whole buffer has been read. To access the next sector, a new LBA address has to be loaded into the address registers.

The write procedure is a little bit more complex. After a write command has been loaded into the command register, an empty data buffer is loaded. This buffer can then be filled by sequentially writing data to it. Before a byte can be written the ready/busy pin of the CF card needs to be checked to verify that the card is ready to accept new data. As soon as the 512-byte buffer has been filled with data, the buffer is automatically transferred to the internal memory of the card. No data can be transferred to the card memory if the data buffer had not been completely filled. To write data to the next sector, the LBA address of that sector needs to be loaded and the data buffer needs to be written again.

#### 3.3.5.4 The FAT32 file system

The operation of a FAT32 file system will be given in detail. This is an important aspect of the software design.

**3.3.5.4.1 Background information** The FAT32 file system has been chosen for this system because it can be read by both Windows and Linux systems. This will enable a user to put a CF card in a card reader, connect it to a PC and drag the files onto the PC. The FAT16 file system was also considered, but it can address only a limited amount of storage space (2 GB) and will not suffice for the large storage cards.

The FAT32 file system consists of the MBR (Master Boot Record), volume ID, FAT tables and the cluster area where the file and directory data are stored. The very first sector on the disk is the MBR. The first 448 bytes of this sector contains boot information and has no use in an imbedded system. This is followed by partition information which holds the starting address of the FAT32 partition (volume ID) and the size of the partition. There is also a file system byte which can be used to test whether the disk is indeed formatted in the FAT32 file system. The LBA address of the volume ID is read from the MBR and the volume ID is accessed. Figure 3.7 shows the structure of a FAT 32 partition. Note that the MBR does not form part of the partition, but simply provides information like the LBA address of the volume ID.

The volume ID contains information about the starting location of the FAT tables, the sector size of the FAT tables, the number of reserved sectors, the cluster size and location of the root directory of the partition. There are a number of reserved sectors after the volume ID and before the start of the FAT tables. The cluster size is given in multiples of the 512 byte sector size and can range from two sectors (1 kB) clusters to 16 sectors (32 kB). Once all the necessary information has been read out of the volume ID, the FAT tables can be accessed. There are two FAT tables so that the operating

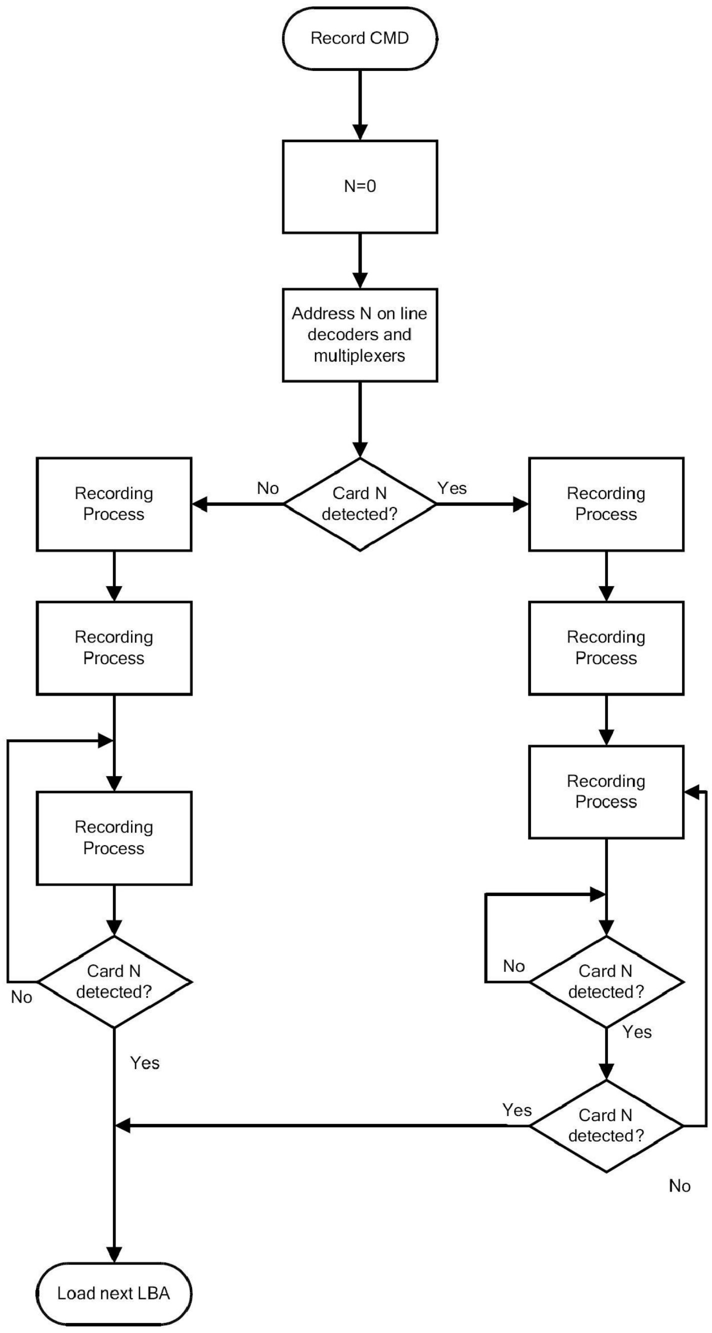


Figure 3.6: A flow diagram describing the data read and write procedure for the CF card.

system of the PC may check for consistency.

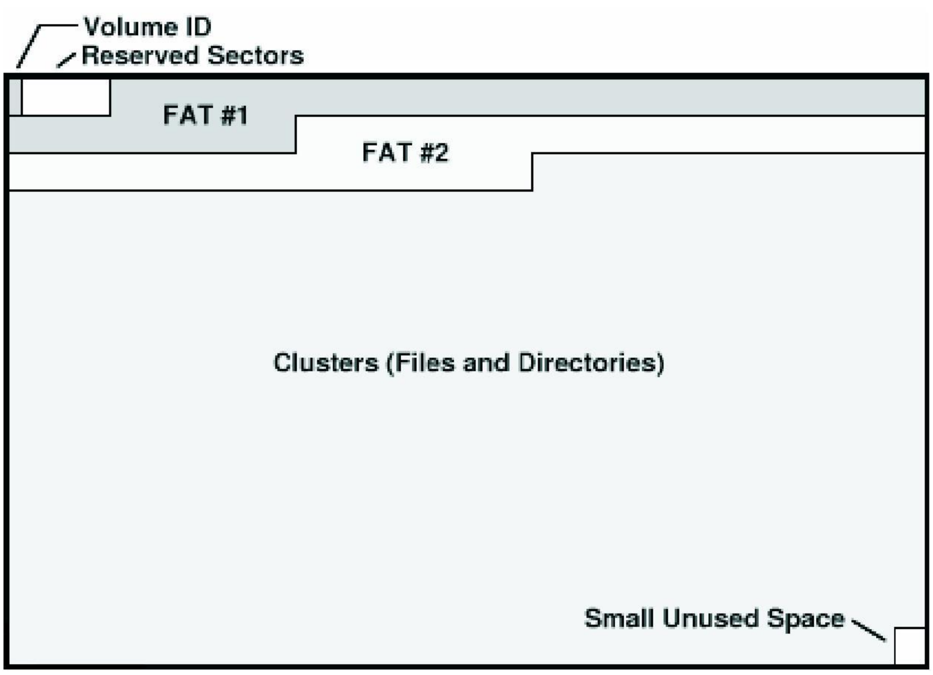
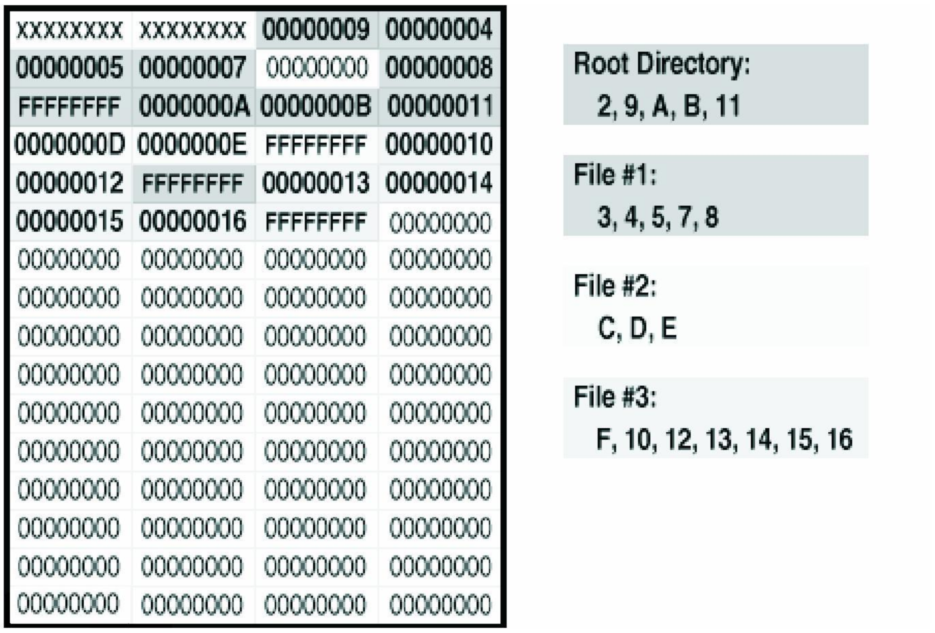


Figure 3.7: A Fat 32 disk partition.

The operation of the FAT tables is straightforward. The FAT table is a map of the cluster area where a four-byte number represents each cluster. This number indicates the location of the next cluster in the chain, or if the bits in that cluster are all set to 1's, it means that the end of the file chain has been reached. Cluster 0 and cluster 1 does not exist, so that the cluster numbering starts at 2. This first cluster is also the start of the root directory. The root directory is the main directory of the partition and all other directories and files are stored under this first directory.

Figure 3.8 shows how a FAT table might look after three small files have been written to the disk. As can be seen from the figure, position 0 and 1 are not applicable and the table starts at position number 2 where the root directory is located. The value stored in position 2 is 9 so the next cluster where the root directory's information will continue is cluster number 9. In cluster number 9, the value of 10 is found so the root directory will continue in cluster 10. This in turn points to cluster 11 and it points to cluster number 17. The bits in position 17 are all set to 1's. This means that the cluster chain ends at this position (the end of the root directory). In a similar fashion, the cluster chains of each of the three files may be followed from their starting point to their finishing point by using the FAT table. The question is now, how does one know at what position of the FAT table a file begins?



| | | | |
|---|---|---|---|
| XXXXXXXX | XXXXXXXX | 00000009 | 00000004 |
| 00000005 | 00000007 | 00000000 | 00000008 |
| FFFFFFFF | 0000000A | 0000000B | 00000011 |
| 0000000D | 0000000E | FFFFFFFF | 00000010 |
| 00000012 | FFFFFFFF | 00000013 | 00000014 |
| 00000015 | 00000016 | FFFFFFFF | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |

Root Directory:
2, 9, A, B, 11

File #1:
3, 4, 5, 7, 8

File #2:
C, D, E

File #3:
F, 10, 12, 13, 14, 15, 16

Figure 3.8: A visual representation of how a FAT table operates, showing the hexadecimal values stored in each position.

The starting cluster number of a file is stored together with the file name entry in a directory. The structure of the 32-byte name entries is shown in Figure 3.9. The short filename consists of an eight character filename with a three character extension code to identify the file type. An example of a file name would be "EXAMPLEFWAV" where the file name is EXAMPLEF and the file type is a wave sound file. The next byte is the attribute byte where the attributes of the file are set. A hexadecimal value of 0x08 sets the archive attribute and other attributes like hidden or read only can also be selected by setting the correct bits.

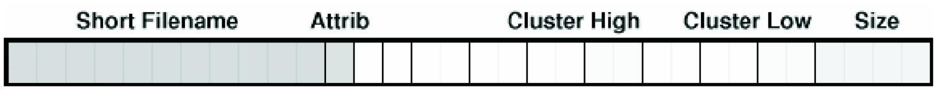


Figure 3.9: The 32-byte directory structure used for short file names in FAT32.

The time and date of file creation and the last time that the file was accessed may also be stored. Four bytes are allocated to store the number of the cluster where the files begin. This value will be used to find the correct position on the FAT table to start the cluster chain for the file. Because a file can end anywhere in a cluster, the cluster information is not enough to correctly determine the file size. For this reason the size of the file (in bytes) is stored in the last four bytes of the file name structure.

The file data is written to the cluster number indicated in its file name structure. If the data is larger than one cluster, the data flows into a next cluster and the FAT tables are updated.

**3.3.5.4.2 Reading FAT32 information** The very first sector on the disk is known as the master boot record. The first 446 bytes of the MBR contain irrelevant information and are skipped. Then follows a 64-byte partition table which is read. This table contains information on the starting location of the partition and what type of file system is used. The type code is tested to determine whether the file system is indeed FAT32. If an invalid code is detected the program will ignore the card and skip to the next card. The starting LBA of the FAT32 partition is read out of the table and the program loads the sector from the CF card.

This first sector of the FAT32 partition, called the volume ID, contains information on the partition, including the number of sectors per cluster, the number of reserved sectors (before the first FAT table), the number of sectors per FAT table and the cluster location of the root directory. The mentioned parameters are the only ones needed for this application. The program flow diagram for retrieving and calculating important FAT32 parameters are given in Figure 3.10.

After all the necessary information has been retrieved, the start of clusters LBA and the FAT tables' begin LBA needs to be calculated by using Equations 3.8 and 3.9.

$$\mathrm{Fat\_Begin\_LBA} = \mathrm{Partition\_LBA\_Begin} + \mathrm{Number\_of\_Reserved\_Sectors}, \quad (3.8)$$

$$\mathrm{Cluster\_Begin\_LBA} = \mathrm{Fat\_Begin\_LBA} + \mathrm{Number\_of\_FATs} * \mathrm{Sectors\_Per\_FAT}. \quad (3.9)$$

With these values calculated, the program now has the ability to manipulate the FAT tables and all the data in the clusters of the FAT32 partition.

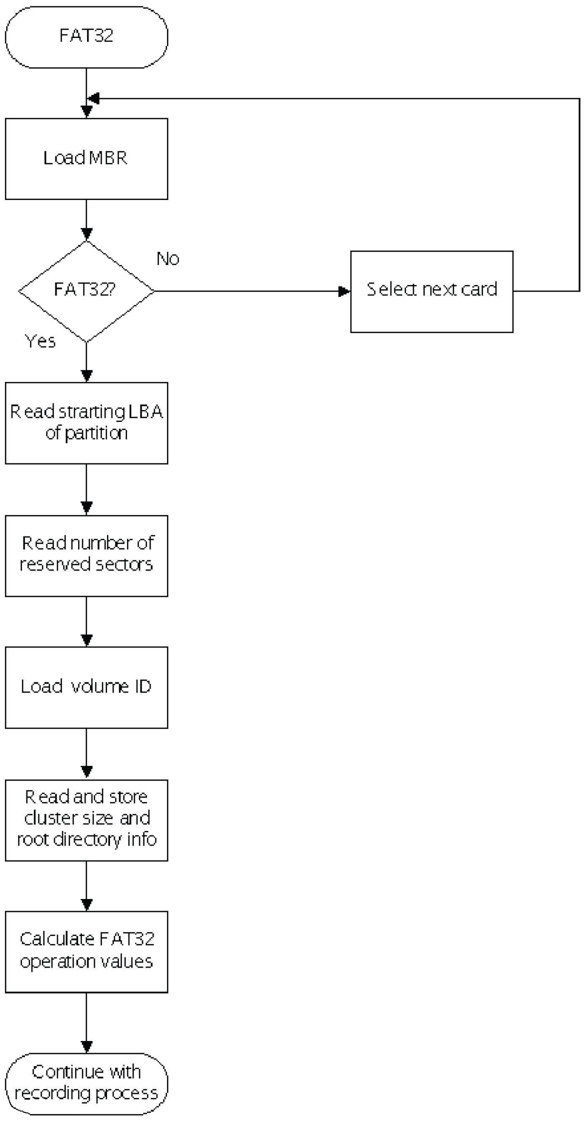

FAT32
Load MBR
FAT32?
No
Yes
Select next card
Read strating LBA of partition
Read number of reserved sectors
Load volume ID
Read and store cluster size and root directory info
Calculate FAT32 operation values
Continue with recording process

Figure 3.10: A flow diagram describing how important FAT32 information is retrieved and calculated.

**3.3.5.4.3 Writing the FAT tables** The FAT table is a map of the cluster area where a four-byte number represents each cluster. This number indicates the location of the next cluster in the chain, or, if the hexadecimal value in that cluster is 0xFFFFFFFF it means that the end of the file chain has been reached. Cluster 0 and cluster 1 does not exist so they have a default value of 0xFFFFFFFF and cluster numbering actually only starts at position 2. Cluster 2 is also the start of the root directory. Each FAT table entry contains the number of the cluster position where the data of the current file continues.

Figure 3.11 shows the flow diagram of how the FAT tables were written. A counter N is used to determine the total cluster position. The counter M is used to determine when the data buffer of the CF card is full so that a new sector may be loaded. Since FAT table entries are 4 bytes in size, 128 table entries fits into one sector. There is also a file size counter to keep track of the size of the current file that is being written to the fat. Finally there is a counter K to determine whether the first or the second FAT table is currently being written to.

All the files written to the cards will have an equal length of Fsize. This part of the program begins by first testing the value of K to determine that both tables have not yet been written to and then writing the position of the next cluster position in each Fat table position. After a value has been written N and F are increased and M is decreased. The values of these counters are all tested after each FAT entry. If M is zero a new LBA address needs to be loaded because the CF's data buffer is full. M is then reset to 128 again. If F equals the file size a hexadecimal value of FFFFFFFF is written to the FAT table to indicate that the file had ended and the value of F is reset to 0. If N is at the capacity of the card (the card can only hold a Ncap amount of clusters) the FAT table have been filled and the next table is written in the same way.

**3.3.5.4.4 Writing the root directory** The root directory is the main directory of the partition and all other directories and files are stored under this first directory.

The file names written in the root directory will be seen in the CF card folder when it is connected to a PC. These files will then be copied by simply dragging them onto the PC. The root directory is written in the way described by the flow diagram of Figure 3.12. The N counter keeps track of the cluster location, where each file starts and increases by the size of a file (in clusters) after every entry and finally causes the

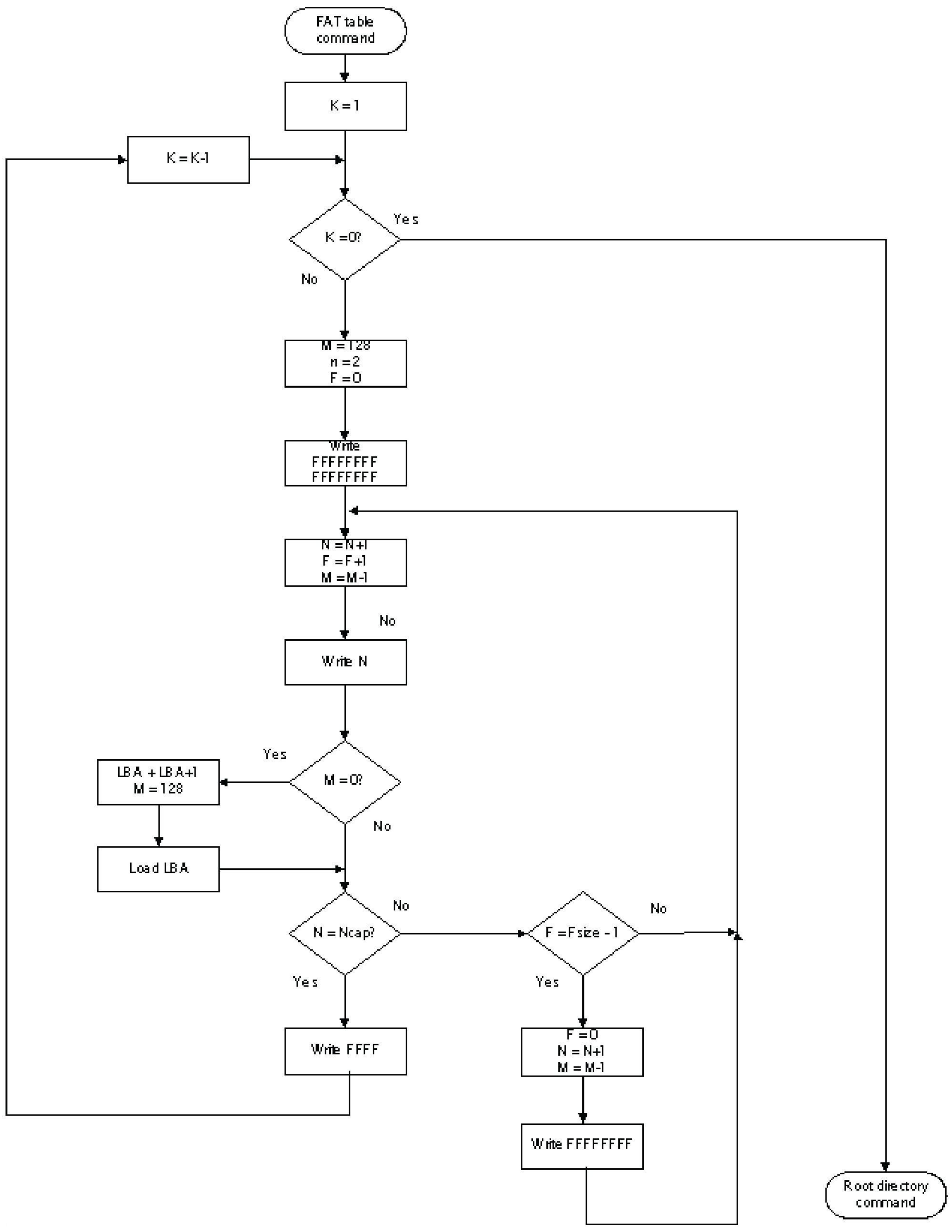


Figure 3.11: A flow diagram describing how the FAT tables are written.

program to go out of the root directory loop when the CF card's capacity has been reached (N equals Ncap). This cluster information is needed so that the operating system of a PC can locate the correct Fat table position for the starting point of the file and follow the cluster chain when copying the file.

The file name structure is 32 bytes in size, which means that 16 directory entries can be made in one 512-byte sector. For this reason the M counter has a starting value of 16 that decreases every time a file name entry has been done. When M reaches zero, the next sector is loaded and the value of M is reset to 16. The Name counter is increased after every entry and has the sole purpose of ensuring that every file name in the root directory is different.

#### 3.3.5.5 Controlling the ADC

The ADC samples a value from the analogue signal from the microphone and converts it to a 16 bit binary number. This binary number needs to be read into two of the microcontroller's registers in order to be processed further. Since the data is output serially on a single pin, a subroutine needed to be written to retrieve the serial data and store it in microcontroller registers. The flow diagram in Figure 3.13 shows how this was achieved.

First, an active low convert pulse is sent to the ADC to initiate a convert cycle. Two clock pulses follow this, the first of which starts before the end of the convert pulse. Sixteen clock pulses are now sent to clock out the data bit by bit. On the rising edge of each clock pulse, the serial output pin from the ADC is read. If the pin is high, the corresponding bit in the ADC data register is set accordingly. If the data pin is low, the corresponding bit in die data register is cleared. The bits are clocked out MSB (most significant bit) first and in the microcontroller two eight bit registers are needed to store the 16-bit binary word.

When storing 16-bit digital sound data in a wave file, the data need to be in the 2's complement form. The last part of the ADC routine converts the binary number to 2's complement by inverting the MSB.

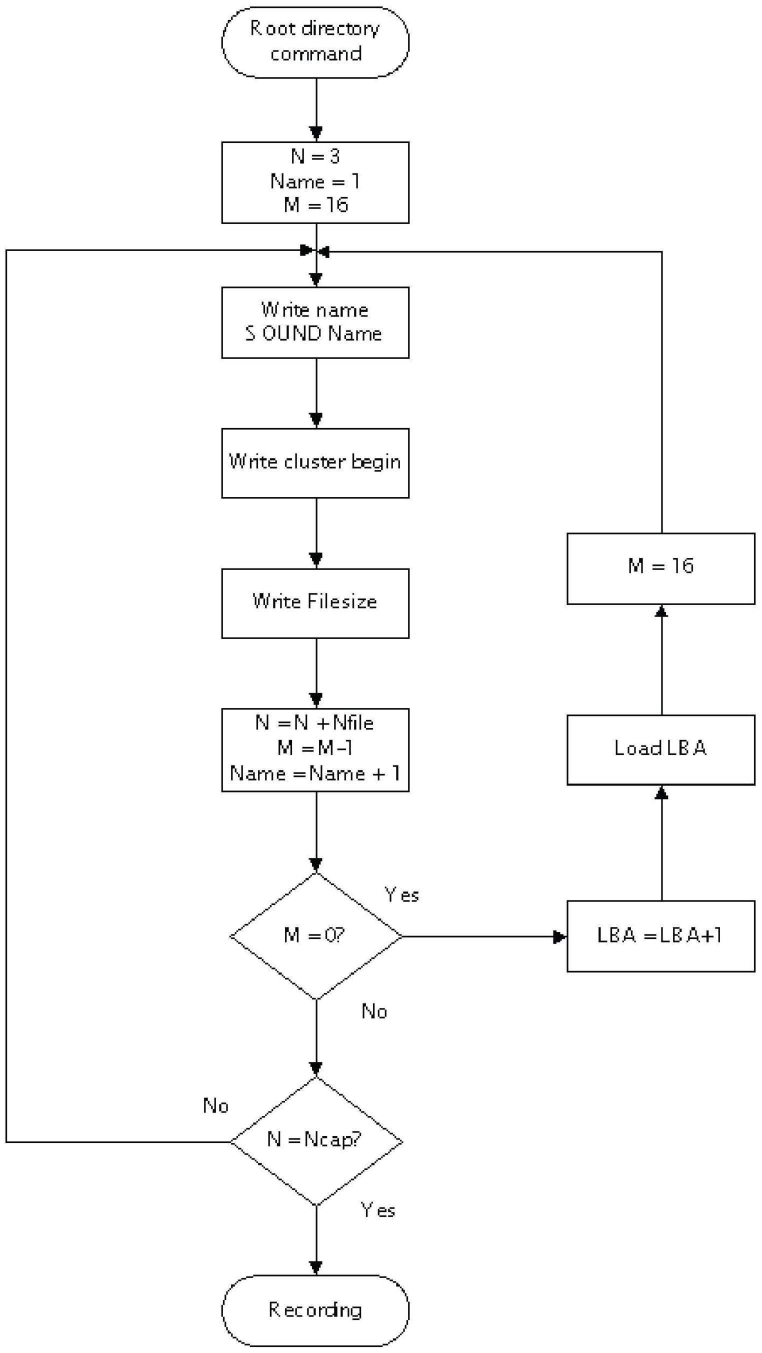


Figure 3.12: A flow diagram describing how the root directory information is written.

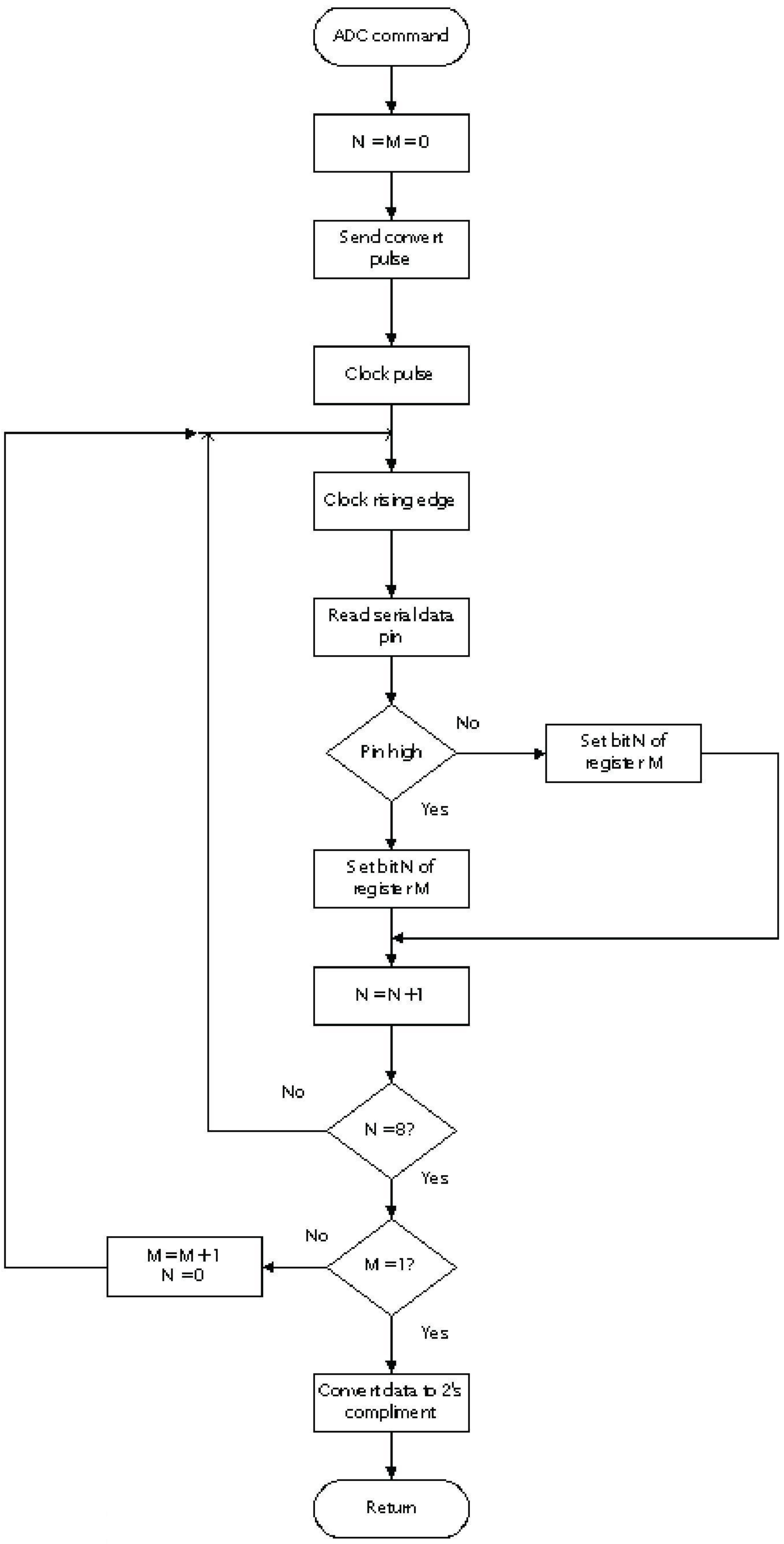


Figure 3.13: A flow diagram describing how the ADC subroutine works.

#### 3.3.5.6 Recording sound

**3.3.5.6.1 The wave file format** The Wave file format is Windows' original file format for storing digital audio data. It has become one of the most widely supported digital audio file formats on PC's because Windows is so popular and a huge number of programs have been written to handle wave files. Almost every modern program that can open or save digital audio supports this file format and it was the logical choice to use it as the file format in which the sound recorded by the designed device will be stored.

Wave files use a RIFF structure which groups the file contents into separate chunks, each containing its own header and data bytes. The chunk header specifies the type and size of the chunk data bytes. Certain types of chunks can contain sub-chunks. In Figure 3.14 it can be seen that the "fmt" and "data" chunks are actually sub-chunks of the "RIFF" chunk.

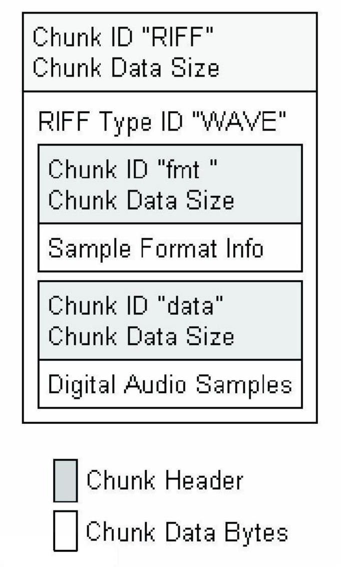


Figure 3.14: The RIFF structure used in the wave file structure.

It is important for correct operation that RIFF chunks should always be word aligned. This means that it should contain an even number of bytes. If this is not the case, an extra byte containing a zero value should be at the end of the chunk.

The RIFF type ID is “WAVE” which identifies this RIFF structure as being a wave file. This is followed by a format chunk containing information like the sample frequency and whether it is mono or stereo sound file. The information stored in the format chunk is displayed in Table 3.3.

Table 3.4: The information bytes stored under the format chunk.

| **Offset** | **Size** | **Description** | **Value** |
|---|---|---|---|
| 0x00 | 4 | Chunk ID | “fmt” (0x666D7420) |
| 0x04 | 4 | Chunk Data Size | 16 + extra format bytes |
| 0x08 | 2 | Compression code | 1 – 65,535 |
| 0x0a | 2 | Number of channels | 1 – 65,535 |
| 0x0c | 4 | Sample rate | 1 – 0xFFFFFFFF |
| 0x10 | 4 | Average bytes per second | 1 – 0xFFFFFFFF |
| 0x14 | 2 | block align | 1 – 65,535 |
| 0x16 | 2 | Significant bits per sample | 2 – 65,535 |
| 0x18 | 2 | Extra format bytes | 0 – 65,535 |

The number of channels specifies how many separate signals that are stored in the wave data chunk. If the number of channels is 1 it is a mono signal, while a value of 2 signifies a stereo signal. The designed system will store a mono channel signal. The sample rate indicates how many times ADC samples of the sound signal are taken in one second. Our system will operate at a sampling rate of 3 kHz. The block align value indicates the number of bytes per sample and can be calculated by using Equation 3.10.

$$\text{BlockAlign} = (\text{BitsPerSample} \div 8) \times \text{NumChannels}. \tag{3.10}$$

The average bytes per second value indicate how many bytes of data must be streamed to the digital to analogue converter of the PC per second in order to play the wave file and may be calculated Equation 3.11.

$$\text{BytesPerSecond} = f_s \times \text{BlockAlign}. \tag{3.11}$$

Bits per sample indicate with how many bits the ADC represents the sample taken from the analogue signal. The elephant recording system has a 16-bit ADC and no extra format bytes will be used.

The data chunk contains the actual digital sound data. When samples are represented with 8-bits, they are specified as unsigned values. All other sample bit-sizes are specified as signed values. For example a 16-bit sample can range from -32,768 to +32,767 with a mid-point (silence) at 0. This implies that the value read from the 16-bit ADC in the elephant recording system should be converted to a 2's complement digital number before being stored.

**3.3.5.6.2 Recording process** After all the file system preparation has been done, the digital sound data may be recorded in sequence with only the wave file format's RIFF structure and time, date, temperature and position in between different files.

The desired sampling frequency of the recording is achieved by triggering an internal interrupt at a frequency of 3 kHz. When this interrupt occurs the program jumps to the interrupt service routine where the ADC flag is set.

As can be seen in Figure 3.15 the main program loops continuously, testing for set flags. When a set ADC flag is detected, the ADC subroutine is called and the new values are stored in the RAM. If the file size is reached, the EOF flag gets set and the program initiates the start of a new file. A new directory entry needs to be written with an unique filename and the FAT tables need to be updated. A temperature measurement and GPS time and coordinates are written and the recording of the next file starts.

When the disk capacity is reached, the capacity flag is set and the recording process will start anew on the next disk. After all disks' capacities have been reached the program goes into a never ending loop waiting for the collar to be retrieved.

#### 3.3.5.7 Thermometer operation

The digital thermometer communicates via a one-wire interface. The I/O pin of the microcontroller that communicates with the thermometer needs to change frequently from an input pin to an output pin to receive data and give commands respectively.

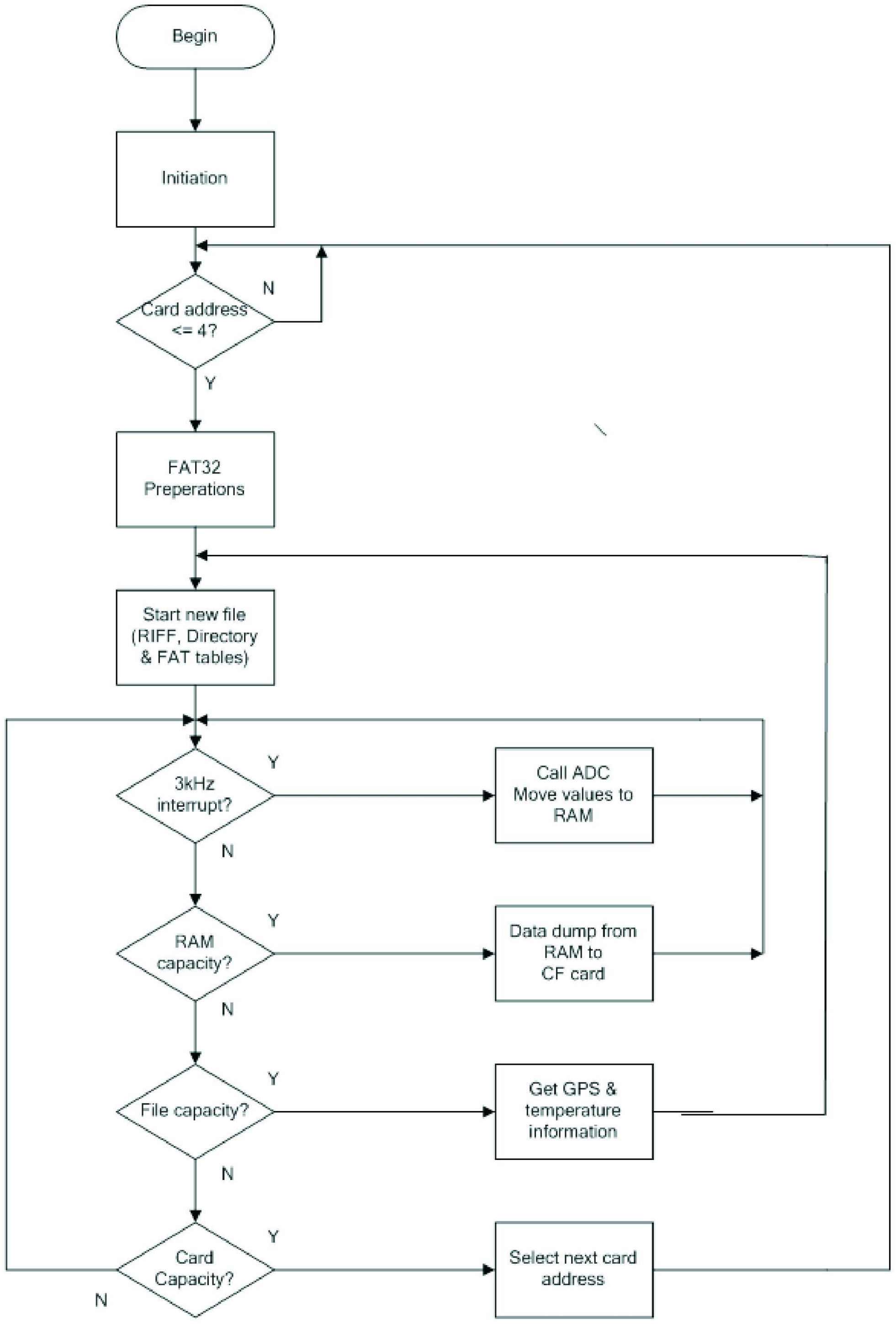


Figure 3.15: The flow diagram on the right describes the interrupt service routine while the diagram on the left is the main program loop reacting on the flags that is set in the interrupt service routine.

Each set of commands begins with a reset pulse after which two instructions are given: CONVERT (convert the outside temperature to a 12-bit digital value) and SEND (send the information) at which point the I/O pin of the microcontroller is set to an input pin to receive the data. Figure 3.16 shows the flow diagram for both the thermometer and the GPS module.

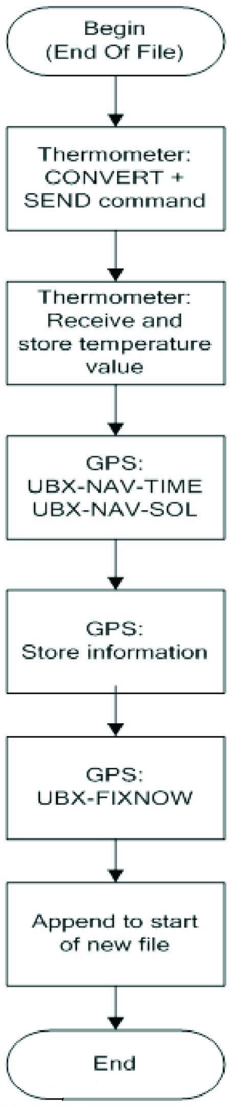


Figure 3.16: A flow diagram describing thesteps in the GPS/Thermometer subroutine.

#### 3.3.5.8 GPS operation

The GPS device is equipped with two USART ports. The PIC16F877 is also equipped with an USART port and communication is established using the RX and TX pins of both devices.

An U-blox customized GPS protocol called UBX is used. An UBX command can be sent to request positional coordinates as well as time and date information (UBX-NAV-SOL and UBX-NAV-TIME respectively). Using the UBX-FIXNOW command the GPS device can be put in a low power mode. As shown in Figure 3.16, the GPS is only waked up for a short while to retrieve time, date and coordinate information and is then put back into a low power mode.

## 3.4 MECHANICAL DESIGN

The objectives with the mechanical design of the collar can be summarized as follows:

1. The memory cards should be easy to access and remove.
2. The electronics should be protected from physical impact.
3. The electronics should be protected from moisture (watertight).
4. When packaged, the microphone should be nearly as sensitive to sound as before packaging (see Chapter 4).
5. The microphone should not be obstructed by mud for extended periods of time (smooth surface above microphone where mud can dry and peel off).

### 3.4.1 Packaging of electronics

An enclosure for the electronics, shown in Figure 3.17, was manufactured. The dashed line represents a cover panel of 5 mm thick polycarbonate. Polycarbonate has excellent mechanical strength properties, but additionally its smooth surface made it an attractive option.

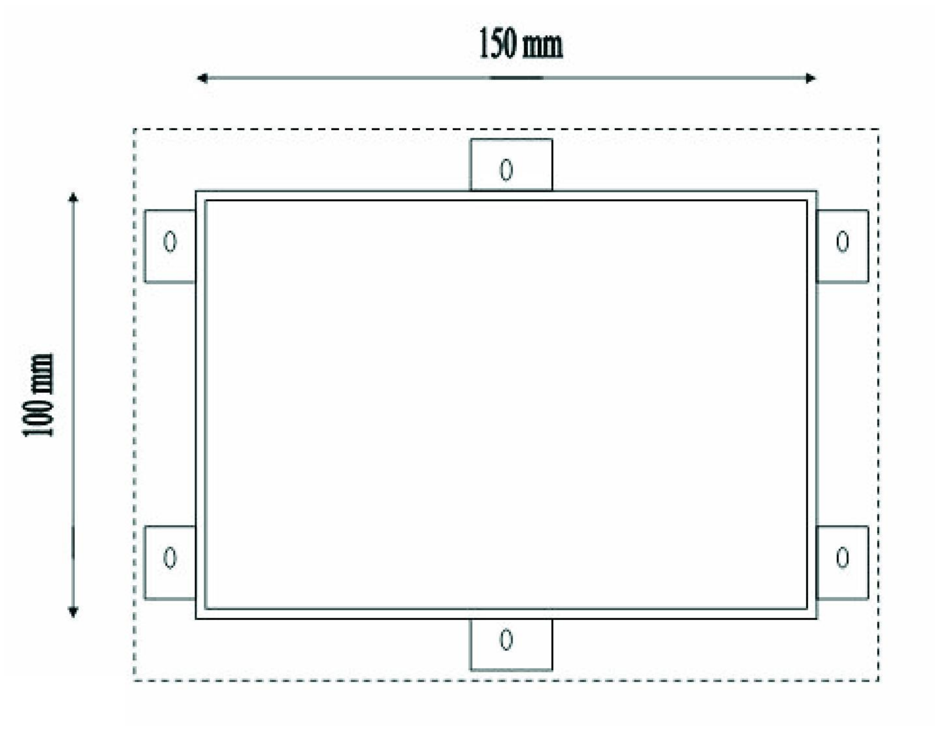


Figure 3.17: Drawing of the enclosure housing the electronics.

The side panels of the enclosure are made of 3 mm Perspex, and six additional Perspex bars were mounted onto the side panels to provide pillars onto which the cover panel can be fastened by screws. Six self-tapping screws are used. Silicon is used to obtain watertight seals all round, including the mounting of the cover panel onto the enclosure. The Perspex pillars also provide a binding surface for the plastic compound that is used to mould the enclosure into the collar.

The electronics boards were encased in silicon within the enclosure to ensure good shock absorbance. The entire enclosure of Figure 3.17 was moulded into the collar with a hard plastic epoxy as shown in Figure 3.18 The plastic epoxy was used instead of dental acrylics, because the acrylics are prone to cracking when rigid objects are moulded within it.

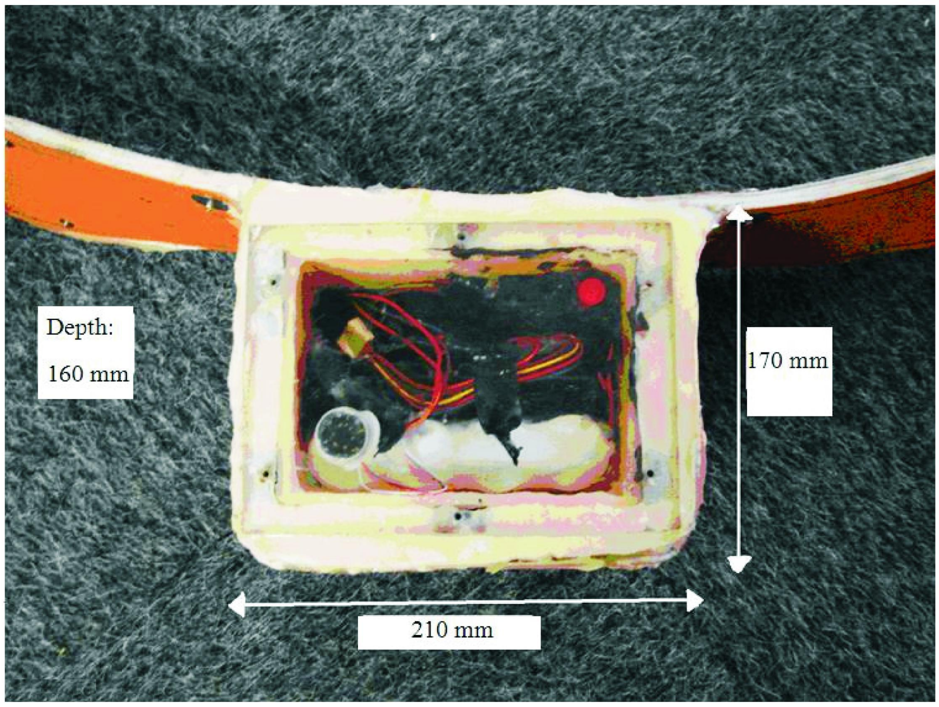


Figure 3.18: A photograph of the collar after the enclosure with the electronics has been mounted into the collar. The weight of the entire collar, including electronics, batteries and packaging is 8.7 kg.

### 3.4.2 Conclusion

First, the packaging of the electronics was completed to achieve the first three objectives given in 3.4.1. Different methods were experimented with to find an optimal way of achieving the last two objectives. These experiments will be discussed in Chapter 4.

## 3.5 AUTOMATIC DETECTION AND PITCH TRACKING OF ELEPHANT RUMBLES

This section discusses the development of an automatic elephant rumble detection algorithm. An analysis of recorded elephant vocalizations are made and elephant rumbles are visually detected to acquire an understanding of the nature of elephant rumbles (3.5.1). The development of the algorithm from speech processing techniques is discussed in Section 3.5.2 and finally the functional steps for the implementation of the algorithm in software are discussed in Section 3.5.3.

### 3.5.1 Analysis of previously recorded data

It was shown in Chapter 2 that elephant vocalizations are identified by experts using the Raven software tool. As a starting point for the development of an automatic elephant rumble detection algorithm, elephant rumbles were first detected visually from previously recorded data. This exercise provided a better insight towards the nature of elephant rumbles which assisted in development of a detection algorithm.

Elephant recordings that had been recorded by a handheld recorder in the Kruger National Park in 2003 by the research group of William Langbauer Jr were provided for use in this study. The recordings were made under a range of different circumstances and in different locations within the southern part of the park. Information about some of the observations that were made while the file were being recorded was also given which helped to locate files that could contain elephant vocalizations. A stepwise method for manually detecting elephant rumbles within recordings are presented in this section.

Raven is a computer program that can be downloaded from the internet (www.brothersoft.com) and is used to draw a spectrogram of a sound file. All the spectrograms shown in this document were created using Raven software.

The exact locations of elephant rumbles within recordings can be identified using the following stepwise method.

1. Open the sound file in Raven to show a spectrogram of the sound.
2. Zoom in to the lower 200 Hz part of the spectrogram.
3. Adjust the program so that 7000 data points are used within the timing window used to calculate the spectrogram. This will ensure that a sharp image of the spectrogram can be seen at low frequencies. A spectrogram now typically appears as shown in Figure 3.19.
4. The spectrogram of a recording with low frequency noise (like the one in the example) may typically appear too bright; the brightness is adjusted to a level at which the information contained in the spectrogram is clearly visible (see Figure 3.20).
5. Areas where low frequency components occurs can be visually identified as horizontal lines on the spectrogram. In this example, a number of areas exist where low frequency energy occurs. The frequency components of both engine noises and elephant rumbles can be observed.
6. The recording is then played back at 10X normal speed. Playing the recording back at faster speeds puts the sounds in the audible range which helps to distinguish between elephant rumbles and other infrasonic sounds such as the engines of motor vehicles which can easily be recognized. In Figure 3.20 the parts of the spectrogram containing the infrasonic elephant rumbles are indicated with red lines.
7. Note that the elephant rumbles has a number of higher harmonics that can be seen as parallel horizontal lines on the spectrogram. This is the main feature that separates the elephant rumbles from other sounds within the infrasonic frequency band.

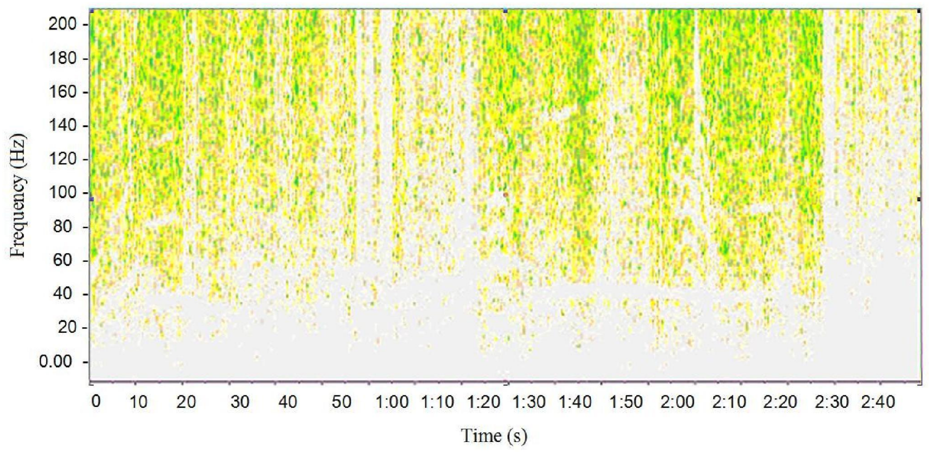


Figure 3.19: The spectrogram of a recording containing subsonic elephant rumbles. This spectrogram is shown at its default brightness.

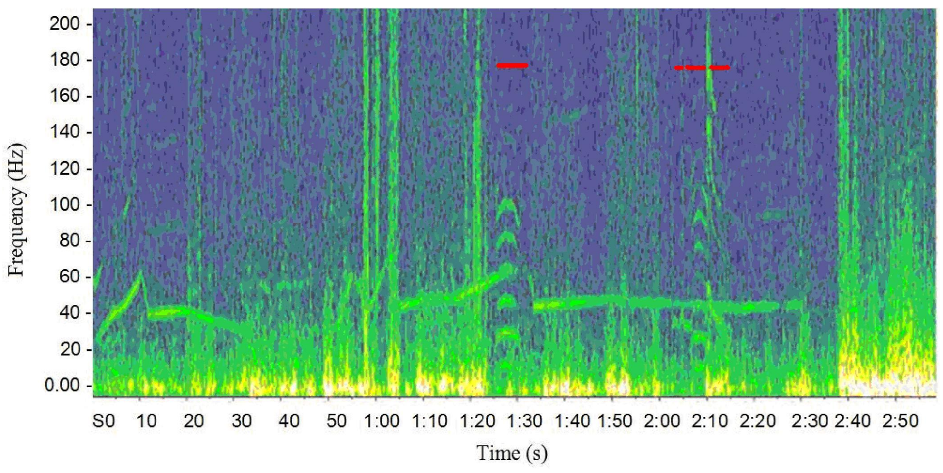


Figure 3.20: The spectrogram of a recording containing subsonic elephant rumbles. The parts of the spectrogram that contains the infrasonic elephant rumbles have been indicated with red lines.

After repeating the procedure mentioned above for a number of different recordings, it became clear that the infrasonic elephant rumbles have some specific characteristics:

1. Rumbles consist of a base frequency with a number of upper harmonics.
2. Rumbles typically have (in the provided recordings) a fundamental frequency of between 14 Hz and 28 Hz.
3. Rumbles have typical durations of between 0.5 to 10 seconds.
4. The majority of rumbles start at a certain pitch that becomes higher in frequency peaking in the middle of the call. The rumble ends at approximately the same pitch that it has started with.

### 3.5.2 Method

The VAD techniques discussed in the Chapter 2 were developed to identify the presence of human speech in a signal. This section will discuss the details of an algorithm devised to automatically detect the presence of infrasonic elephant rumbles. The proposed algorithm uses some of principles of the model by Wu *et al.* (2003), but some of the techniques have been modified and implemented differently. The final stage, where pitch tracks are determined and the final estimates of the rumble locations are calculated makes use of alternative methods altogether.

Figure 3.21 shows the schematic diagram of the proposed signal processing. A single channel input sound enters the system. This sound is divided into different channels by the filter bank. The sound channels are then windowed and the normalized autocorrelation of each window is calculated. Channels with the best signal-to-noise ration are then selected and these are added together to form a final correlogram with which the pitch of that particular time window is estimated. Finally, all the pitch information for every time window is analysed by the last block to form pitch tracks and estimate the location of possible rumbles.

A detailed description of each of the signal processing stages is given below.

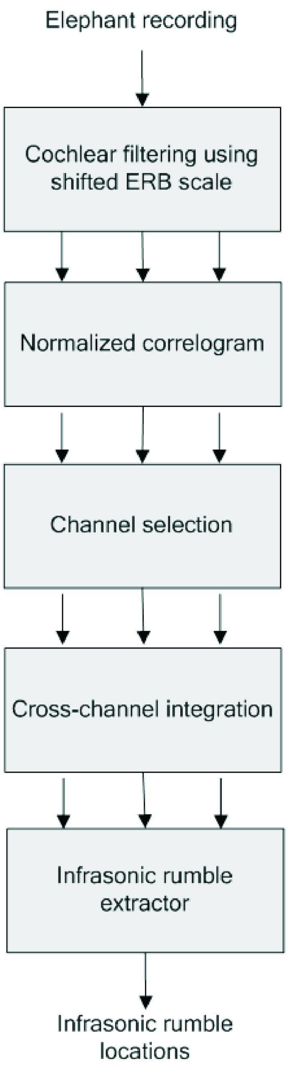

Elephant recording
Cochlear filtering using shifted ERB scale
Normalized correlogram
Channel selection
Cross-channel integration
Infrasonic rumble extractor
Infrasonic rumble locations

Figure 3.21: The schematic diagram of the adapted model used for elephant rumble identification

#### 3.5.2.1 Input

The input to the system will be single channel recorded sound data from either hand-held recorders or elephant collars.

#### 3.5.2.2 Filter bank

The sound enters the filter bank where it is then filtered into sub-bands. Cochlear filtering, a bank of fourth-order gammatone filters evenly distributed on the ERB (Equivalent Rectangular Bandwidth) scale, has been proposed if human speech had to be identified. The use of gammatone filters is a sensible choice where the determination of pitch is concerned since the particular sensitivity of the human cochlea to pitch was used as a template for the development of this filter. The pitch of human speech is typically close to 100 Hz for a males and 200 Hz for females, while elephant rumbles have pitches of between 14 Hz and 25 Hz. An intuitive modification in designing an appropriate filter bank for elephant vocalizations was to use a shifted ERB scale. The ERB (Equivalent Rectangular Bandwidth) scale was shifted by a factor of 10 and pilot tests showed that the rumble detection algorithm performed better than when using the normal human ERB scale. A bank of 32 filters were used with centre frequencies evenly distributed on the shifted ERB scale between 12 Hz and 30 Hz. Figure 3.22 shows the filter bank with the 32 gammatone filters evenly distributed on the shifted ERB scale. The figure was generated in Matlab. Gammatone filters are defined in Katsiamis, Drakakis and Lyon (2007), but for the present algorithm the implementation of the filters was done using the appropriate function from the Matlab Signal Processing Toolbox.

#### 3.5.2.3 Normalized autocorrelation

Autocorrelation is a statistical concept that expresses the degree to which the value of an attribute at spatially adjacent points varies with the time separating the observations. The normalized autocorrelation of each of the channels, *c*, is calculated using Equation 3.12, taken from Wu *et al.* (2003), where N is the number of samples used for the calculation of the autocorrelation. A window size of 40 ms was used, which corresponds to 128 samples in a signal sampled at 3 kHz. The choice of N

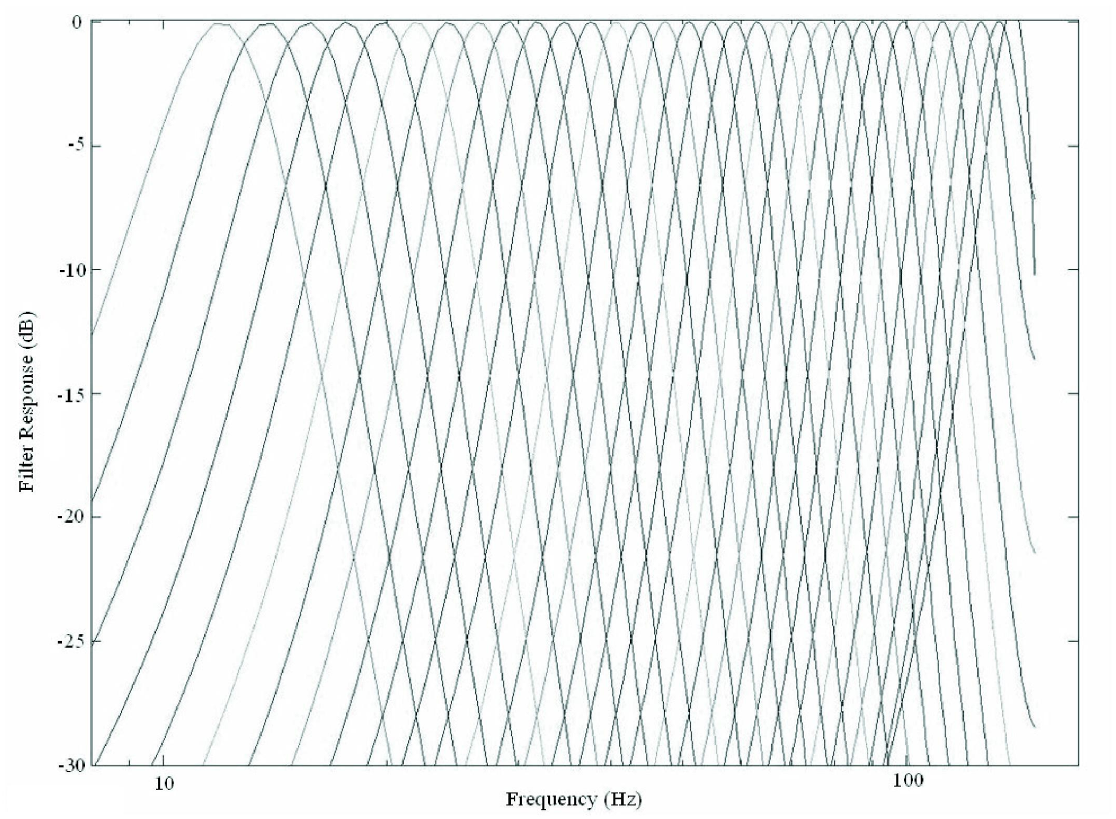


Figure 3.22: Matlab generated response of a filter bank consisting of 32 gammatone filters evenly distributed on the shifted ERB scale.

determines the resolution of the processed data. The output of the filter is denoted as $r(\mathrm{t})$. The number of lag steps, $T$ , used within the autocorrelation determines the lowest frequency that may be detected and needs to be at least 300 samples to detect a frequency of 10 Hz. The position in the centre of the window is denoted as $j$. The autocorrelation is then

$$A(c,j,\tau)=\frac{\sum_{n=-N/2}^{N/2} r(c,j+n)\, r(c,j+n+\tau)}{\sqrt{\sum_{n=-N/2}^{N/2} r(c,j+n)}\sqrt{\sum_{n=-N/2}^{N/2} r^{2}(c,j+n+\tau)}}. \tag{3.12}$$

The numerator is the autocorrelation, while normalization is performed by the denominator.

The correlogram of a channel containing a pure tone is shown in Figure 3.23. It can be seen from the figure that the correlogram of a pure tone is also a periodic waveform with amplitude of one. The number of lag steps at which the first positive peak is located can be used to determine the frequency of the input signal using Equation 3.13.

$$f=\frac{F_s}{p}, \tag{3.13}$$

with $F_s$ the sampling frequency of the input signal and $p$ the number of lag steps before the first positive peak occurs. This equation shows that as the samples used in the autocorrelation are shifted over each other, a maximum peak will occur when the original signal and the shifted signal overlaps.

#### 3.5.2.4 Channel selection

This subsection will explain how the amplitude of the correlograms of each sub band is used to select only sub bands with a good SNR. It should be remembered that the filter bank creating the sub bands are spaced close enough that only a single harmonic component of a harmonic sound can be present in any given sub band.

Figure 3.23 illustrated that a perfectly sinusoidal signal produces a normalised auto-correlogram with peak values of 1. A correlogram of a pure tone signal with added white Gaussian noise was generated in Matlab and is shown in Figure 3.24. This figure shows that the correlogram is still periodic, but with a lower amplitude. As more noise

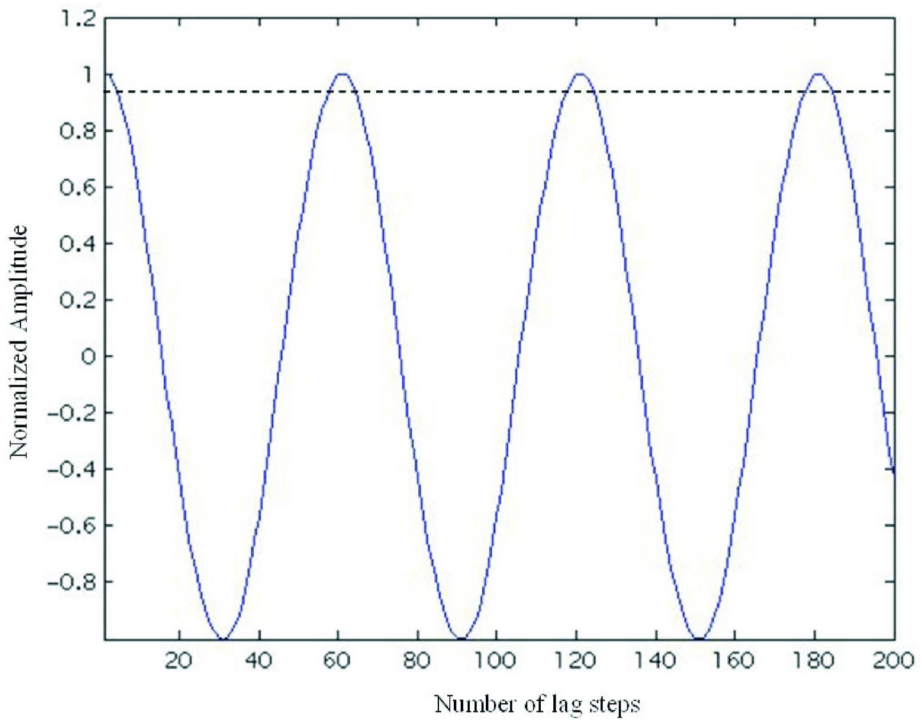


Figure 3.23: The correlogram of a pure tone signal.

is added to the signal, the amplitude of the waveform in the correlogram decreases. The correlogram of a signal consisting of white Gaussian noise is shown in Figure 3.25. It can be seen that the waveform in this figure is non-periodic and at no point has amplitude of more than 0.2 which demonstrates the relationship between the peak values of the correlogram and the periodicity of the applied signal.

The autocorrelation calculation is performed on each of the channels originating from the filter bank. The amplitude of the first positive peak in the correlogram of each channel gives an indication of the frequency component present in that channel and the maximum amplitude of the positive peaks in the correlogram gives an indication of the amount of noise present in the channel for the analyzed segment. Only channels with maximum normalized amplitude of more than 0.945 is selected (this threshold was established empirically in Wu *et al.*, 2003). This method of channel selection enables the algorithm to detect faint calls in recordings with severe broadband disturbances, since only channels with energy at specific pure frequencies are selected regardless of the strength of the component.

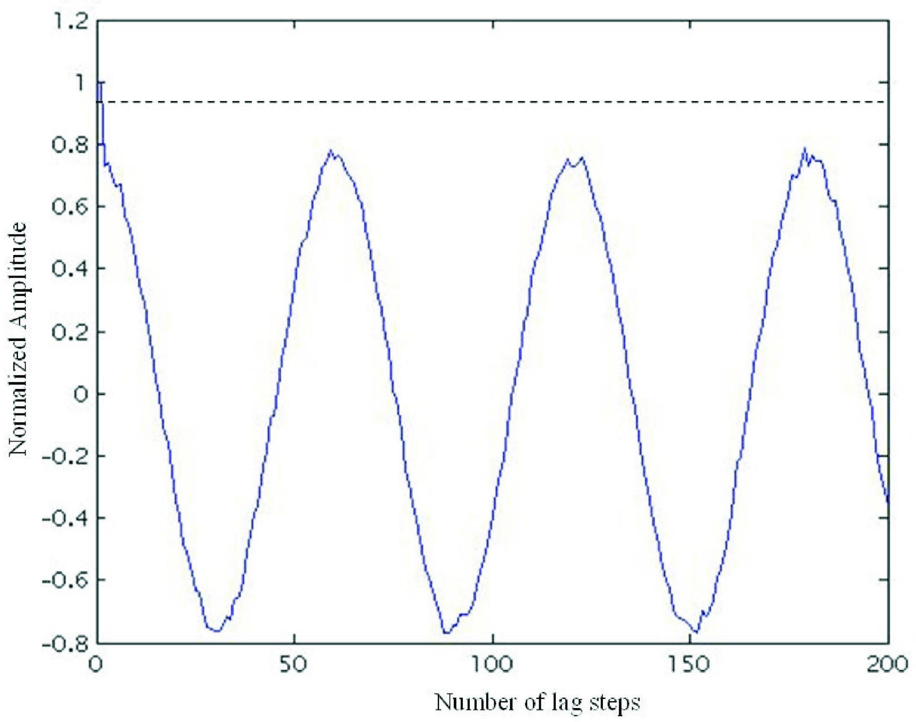


Figure 3.24: The correlogram of a pure tone signal with some white Gaussian noise added.

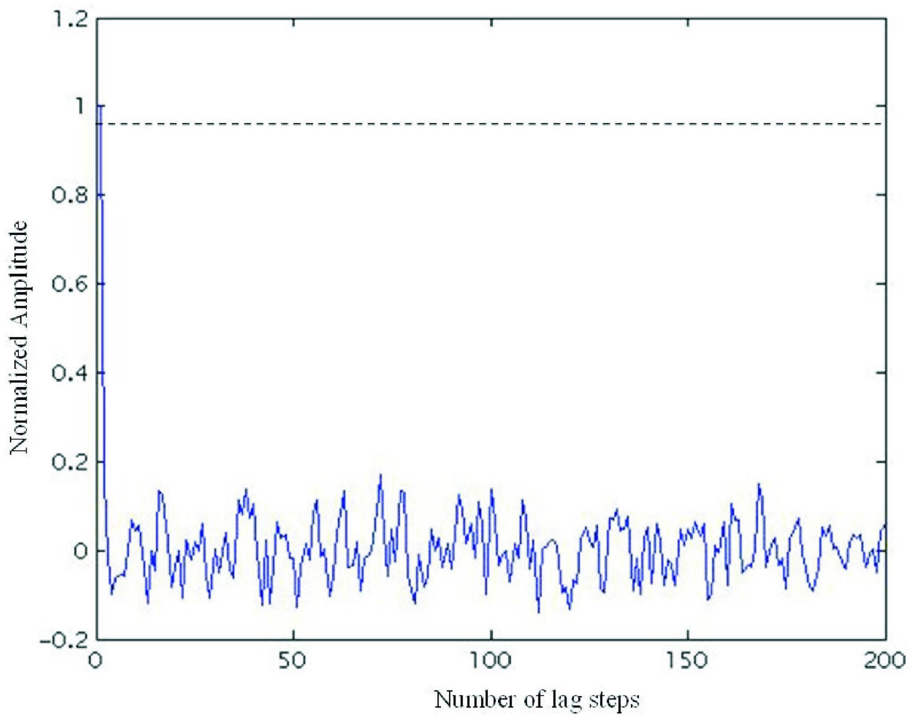


Figure 3.25: The correlogram of white Gaussian noise signal.

### 3.5.2.5 Channel integration and determination of pitch information

When a segment of sound is analyzed, the sound is divided into sub-bands by the filter bank. Channels with low SNR are selected and all the selected channels are added together to form a summated correlogram. The largest positive peak in this summated correlogram is selected, and the corresponding pitch is calculated using Equation 3.13. The way in which this operation detects the pitch of a harmonic signal will be explained with the aid of graphs that have been created in Matlab.

Figure 3.26 shows the spectrogram of a segment of sound with a pitch of 25 Hz consisting of the fundamental frequency of 25 Hz and two harmonic frequencies at 50 Hz and 75 Hz. All the harmonics in the sound possess equal energy.

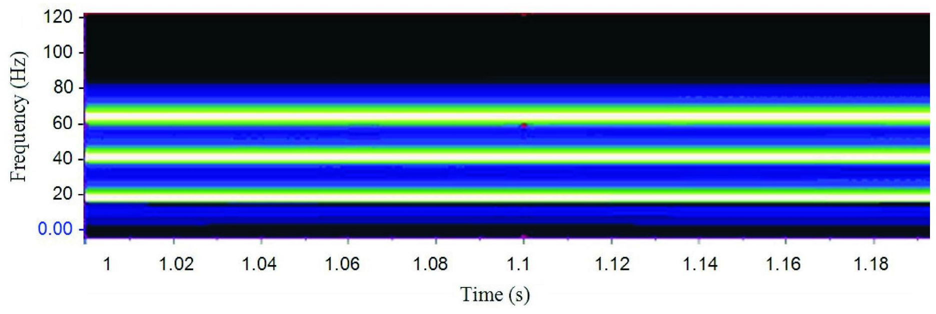


Figure 3.26: Spectrogram of a sound consisting of a 25-Hz fundamental frequency and two harmonics at 50-Hz and 75-Hz.

Figure 3.27(a) shows the normalized correlogram for a channel containing the 25 Hz signal. It can be seen that the first non-zero peak frequency occurs at 120 lag steps, which corresponds to 25 Hz when using Equation 3.13. In the same way Figure 3.27(b) shows the correlogram of a channel containing the 50 Hz signal. In this case the first non-zero peak frequency occurs at 60 lag steps, which correspond to 50 Hz. Figure 3.27(c) shows the normalized correlogram for a channel containing the 75 Hz signal. It can be seen that the first non-zero peak frequency occurs at 40 lag steps which corresponds to 75 Hz according to Equation 3.13. Figure 3.27(d) shows the summed correlogram obtained by summating the correlograms of the 25 Hz, 50 Hz and 75 Hz signals. This figure shows that the positive peak in the summed correlogram with the highest amplitude occurs at 120 lag steps. Equation 3.13 estimates the pitch of the analyzed sound segment as 25 Hz. As can be seen from this example, the presence of higher harmonics in a sound signal assists to strengthen the peak in the summed correlogram that indicates the pitch of the sound.

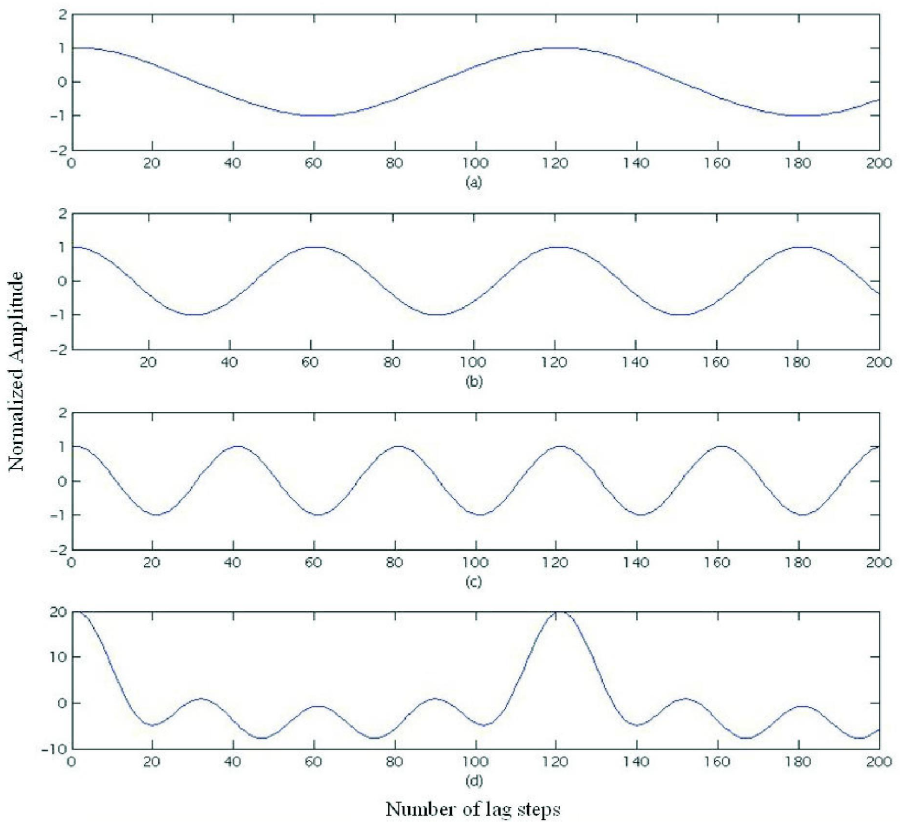


Figure 3.27: (a) Correlogram of a channel containing the 25-Hz fundamental frequency; (b) Correlogram of a channel containing the 50-Hz harmonic frequency; (c) Correlogram of a channel containing the 75-Hz harmonic frequency; and (d) Summed correlogram with the maximum positive peak indicating the pitch of the sound.

Figure 3.28 shows the spectrogram of a segment of sound with a pitch of 25 Hz consisting of only the first three harmonics namely 50 Hz, 75 Hz and 100 Hz.

Figure 3.29(a) shows the normalized correlogram for a channel containing the 50 Hz signal. It can be seen that the first non-zero peak frequency occurs at 60 lag steps, which corresponds to 50 Hz when using Equation 3.13. In the same way Figure 3.29(b) shows the correlogram of a channel containing the 75 Hz signal. In this case the first non-zero peak frequency occurs at 40 lag steps which correspond to 75 Hz. Figure 3.29(c) shows the normalized correlogram for a channel containing the 100 Hz signal. It can be seen that the first non-zero peak frequency occurs at 30 lag steps,

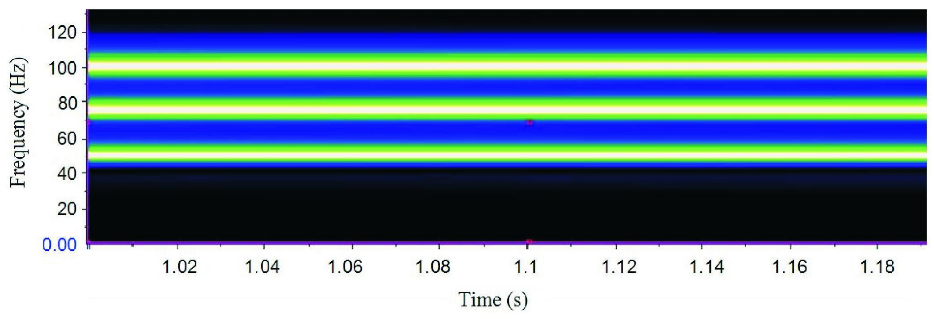


Figure 3.28: Spectrogram of a sound with 25-Hz pitch but with only the first three harmonics present while the fundamental frequency is not present.

which corresponds to 100 Hz according to Equation 3.13. Figure 3.29(d) shows the summed correlogram obtained by summating the correlograms of the 50 Hz, 75 Hz and 100 Hz signals. As can be seen in the figure, the positive peak in the summed correlogram with the highest amplitude occurs at 120 lag steps. The pitch of the analyzed sound segment can be calculated as 25 Hz using Equation 3.13. Note that the correct pitch can still be determined from the summed correlogram even though the fundamental frequency is not present in the signal.

#### 3.5.2.6 Determination of rumble locations and tracking of pitch

Consider the segment of sound shown in the spectrogram of Figure 3.30. It can be seen that an elephant rumble occurs from around 2 seconds up to 5.5 seconds of the recording. After performing the signal processing steps on this sound segment, the pitch information shown in Figure 3.31 is calculated from the maximum peak in each of the summed correlograms. It can be seen from the figure that this pitch information resembles a noisy signal in the area where the rumble is not present, but a smoother pitch track in the area where the rumble occurs. There is however some points of the pitch information that does not follow the pitch track.

The final step of the algorithm should be to extract only the pitch tracks of elephant rumbles while ignoring all the other pitch information. Some additional processing needs to be done to achieve this. When comparing Figure 3.30 and Figure 3.31, note that the section(s) of the estimated pitch where rumbles are present, can be clearly seen as a smooth pitch track, containing perhaps some discontinuities. After

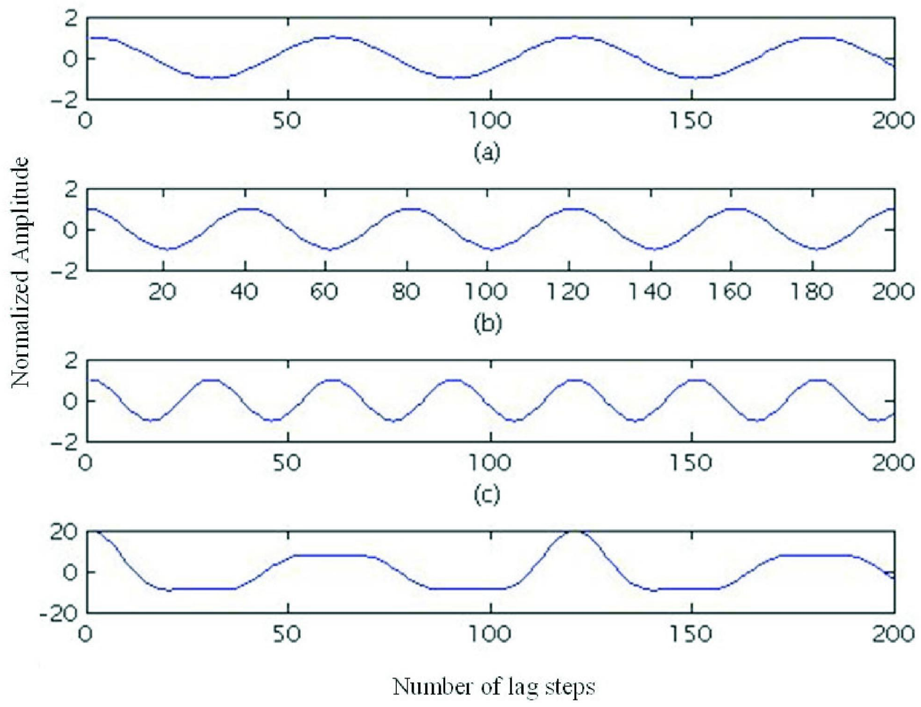


Figure 3.29: (a) Correlogram of a channel containing the 50-Hz harmonic frequency; (b) Correlogram of a channel containing the 75-Hz harmonic frequency; (c) Correlogram of a channel containing the 100-Hz harmonic frequency; and (d) Summed correlogram with the maximum positive peak indicating the pitch of the sound.

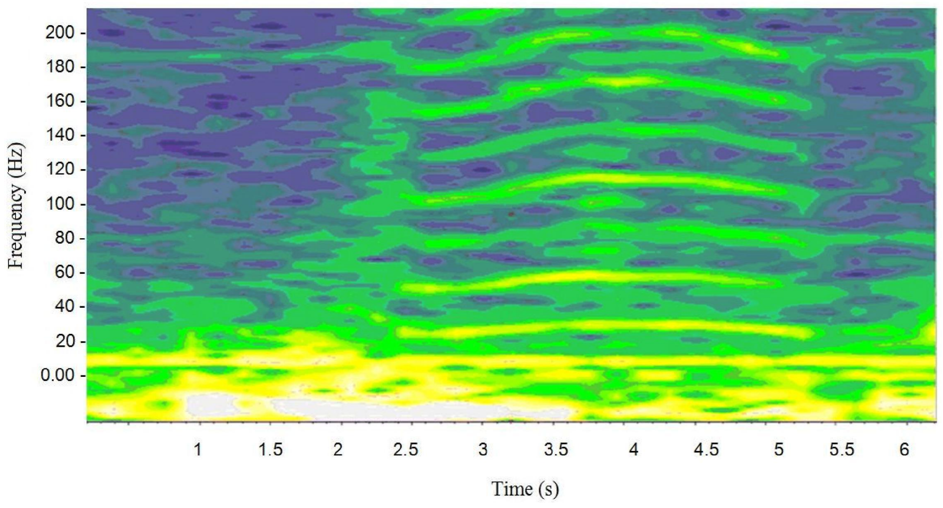


Figure 3.30: Spectrogram of a sound segment containing an infrasonic elephant rumble.

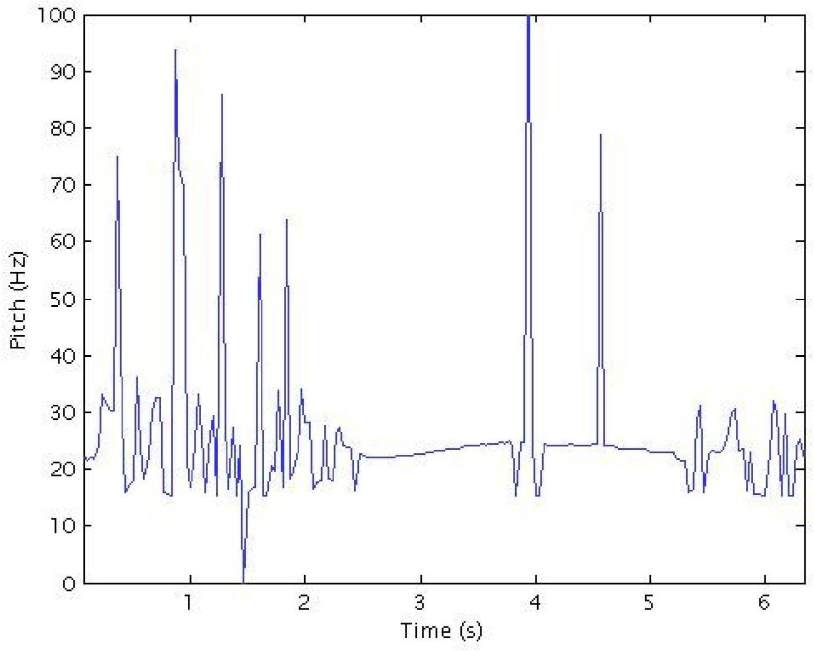


Figure 3.31: Pitch information as calculated for the sound segment shown in Figure 3.34 before pitch track extraction algorithm have been run.

verifying these results with a number of clearly defined elephant rumbles it appears to be a simple matter to determine the locations of the elephant rumbles from the pitch information array by visual inspection of the pitch array.

A number of techniques may be considered to identify the location of elephant rumbles automatically. These include computationally expensive techniques involving the use of HMMs (Paris and Jauffret, 2003; Xie and Evans, 1993) and neural networks (Adams and Evans, 1994).

A simple algorithm (with light computational demands) that mimics the way that one would identify the calls visually were developed. Figure 3.32 describes this algorithm. The algorithm scans through the pitch information array (created in Matlab), looking for smooth parts that indicates the presence of a rumble. If two successive samples differ with less than $S$ Hz it is considered "smooth". If a number of $Pf$ smooth samples occur consecutively, that section of the pitch information array is marked as a pitch track. If a smooth pitch track ends with a discontinuity it is tested whether there is evidence of the same the pitch existing track further on in the pitch information array.

The algorithm will look ahead for $F$ samples in a "beam" with a width defined by $\theta$ as shown in the figure. If another point of the pitch track is discovered in that area, it is assumed that the discontinuity can be ignored and the pitch track is extended to that point. When no further evidence of the pitch track can be found, the location of the pitch track is stored and the algorithm continues searching through the pitch information array as before.

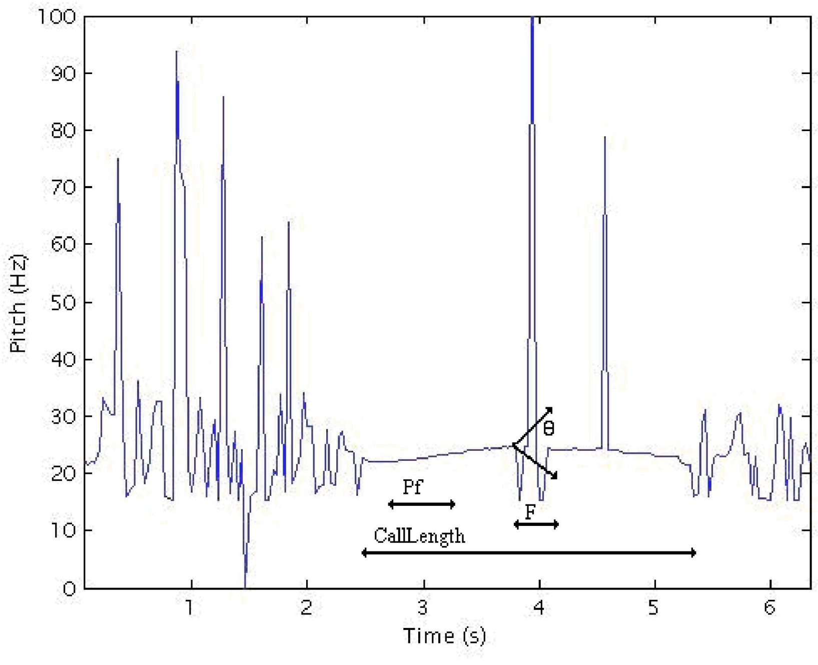


Figure 3.32: Parameters of the pitch track extraction algorithm.

This process is repeated backwards (from the last point of the pitch information array to the first point) to ensure that complete pitch tracks are identified. The pitch tracks from the forward and backward sweeps are combined to estimate the locations of the infrasonic elephant rumbles. Figure 3.33 shows the final output generated after the pitch information array has been processed by the algorithm. In this example, a near perfect estimate of the location of the rumble was obtained.

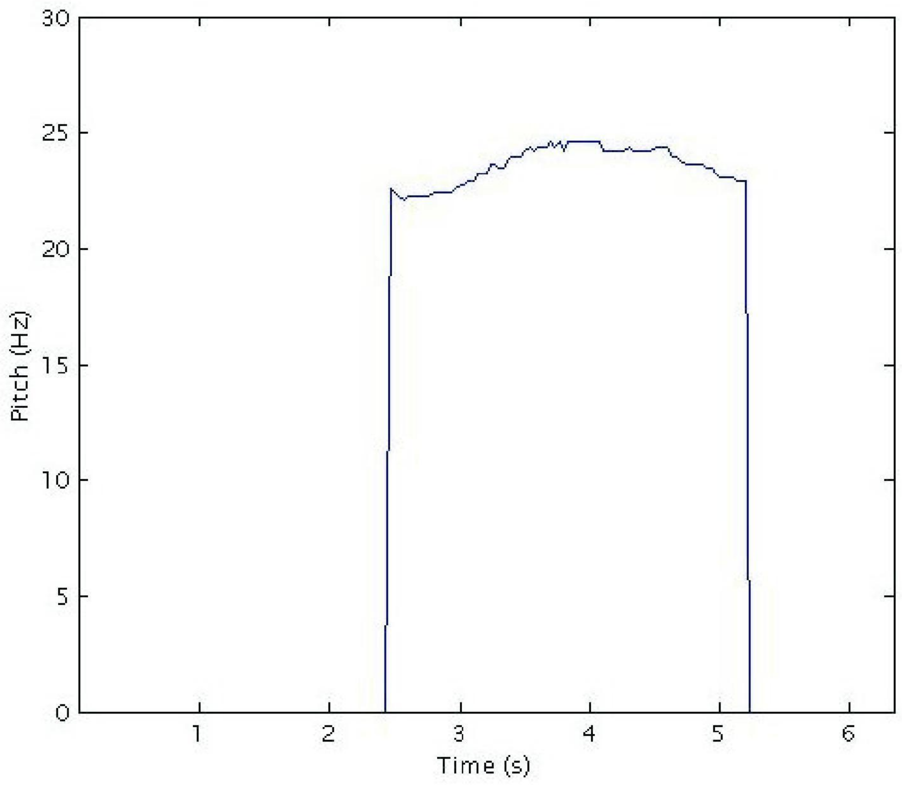


Figure 3.33: Final output of the system when the sound segment shown in Figure 3.34 is analyzed.

### 3.5.3 Implementing the algorithm in program code

Up to this point, the discussion of the algorithm was from a theoretical viewpoint. Some practical problems needed to be solved when implementing the algorithm in software. The elephant rumble detector and pitch tracking algorithm was implemented in the Matlab programming language. Figure 3.34 shows the basic block diagram describing how the algorithm was implemented in software. A functional description (not related to a specific programming language) of the implementation of the algorithm in software will be given.

Firstly, the wave file containing the recorded sound has to be read into memory. If the sampling frequency of the file is different from 3 kHz it needs to be re-sampled to a rate of 3 kHz. It is important to realize that the single channel sound input is divided into 32 (or more) channels at the filter bank. Each of these channels needs as much memory space as the single channel wave file. If the wave file is relatively large (exactly when a file will be too large depends on the resources available on the specific computer on which the algorithm needs to be implemented), memory management needs to be done to ensure that the estimation process runs at optimal speed and so that the computer does not run out of RAM memory.

The implementation of the memory management is shown in the block diagram of Figure 3.35. A buffer length of *BufferSize* is defined by the user based on the amount of RAM that the computer has access to. If the wave file is smaller than *BufferSize* no memory management is done and the whole file is processed at once. If the wave file is larger than *BufferSize* only the first *BufferSize* samples of the original wave file is processed and the program returns to the starting point of this diagram to process the next segment of the wave file. As soon as the remaining number of samples that needs to be processed is less than the size of the buffer, the value of *LastBuffer* is set to 1 to let the program know that it is now processing the last buffer and it will not be necessary to return to this point of the program again. Managing the computer's memory in this way ensures that the performance of the computer will be the same for large files as for small files.

The filter bank consists of a number of fourth order gammatone filters with centre frequencies distributed evenly on the shifted ERB scale as discussed in the previous section. These filters are implemented by using the Auditory Toolbox in Matlab. This

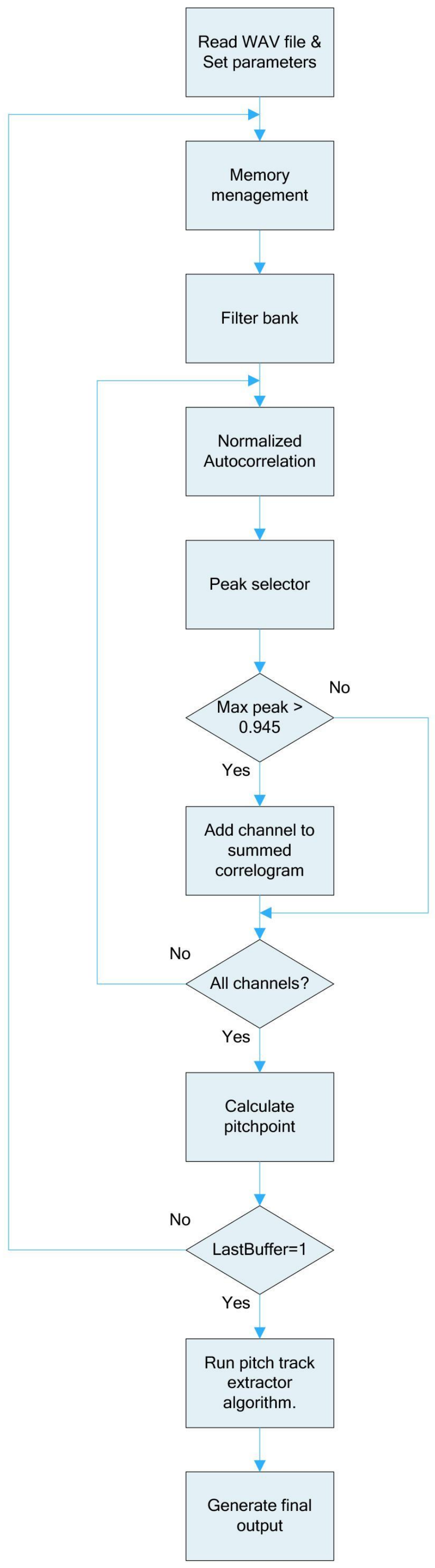


Figure 3.34: Block diagram showing the basic implementation of the algorithm.

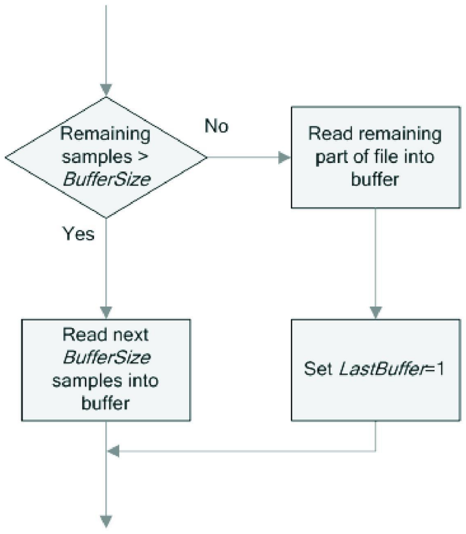

Remaining samples > *BufferSize*
No
Read remaining part of file into buffer
Yes
Read next *BufferSize* samples into buffer
Set *LastBuffer*=1

Figure 3.35: Block diagram showing the operation of the memory management stage of the system.

toolbox can be run an any version of Matlab and can be downloaded from the internet without cost. The centre frequencies range from 12 Hz to 150 Hz and 32 filters are used.

Each of the channels resulting from the filter bank is windowed into smaller segments. The windowed segments have a size of 100 samples (corresponding to 33.33 ms). The normalized autocorrelation function given in Equation 3.12 is performed on each windowed segment of each channel. The autocorrelation is done over 128 samples. The number of lag steps calculated in each autocorrelation is 250. According to Equation 3.13, this means that a pitch as low as 12 Hz can be detected. The window length, the number of samples over which the autocorrelation is done and the number of lags steps to be calculated are defined as parameters in the program and can easily be changed by the user of the program.

The previous section indicated how it is determined whether a channel is noisy or clean. The positive peaks of the correlogram are tested and if the peak has amplitude of more than 0.945 it is considered noise-free. A peak detection algorithm was developed to analyze each correlogram. The pseudo code for the peak detector is shown in Figure 3.36.

```
n=1
If n<EndOfFile
        State=Falling
        If C(n)>C(n+1)
        Begin
                If Sate=Rising
                Begin
                        Peak(a)=n
                        State=Falling
                End
        End
        If C(n)<C(n+1)
        Begin
                If Sate=Falling
                Begin
                        State=Rising
                End
        End
End
```

Figure 3.36: Pseudo code showing the operation of the peak detection stage of the system.

The autocorrelation array is imported into the peak detector. Since the normalized autocorrelation always has amplitude of 1 at lag number 0, the autocorrelation curve will initially decrease monotonically at larger lag steps. The difference between successive samples is calculated. If the result is positive and the values are decreasing, it is known that the current position of the marker is still on the downward slope of a peak. If, however, the result is negative while on a downward slope, it is known that a valley has been reached. The state is now changed to rising and the point is marked as the starting point of a positive peak. If another positive peak is reached in this correlogram, this point is also marked as the end of the previous positive peak segment. The difference between the next two samples is now calculated. If the result is negative and the state is increasing, we know that we are still on the upward slope of a peak. If, however, the result is positive while the state is increasing we know that a positive peak has been reached. The state is now changed to falling and the point is marked as the location of a positive peak. The amplitude at this point is recorded (to establish whether the channel is noisy or not). An array with all the peak information stored is sent back to the main program. Using this information, the algorithm decides whether or not to add the channel to the summed correlogram for that specific time window.

The position of the maximum point in the summed correlogram gives the value of the most prominent pitch present in the sound window. This pitch information point is added to the pitch information array. This process repeats itself until the complete memory buffer has been processed. A new memory buffer is then processed and added to the pitch information array. When all the information in the sound file has been processed a complete pitch information array is available.

The theory behind the technique used for identifying infrasonic elephant rumbles from the pitch information array was discussed in the previous section. Figure 3.37 shows the pseudo code for the rumble extractor. Firstly the pitch information array is imported. The algorithm starts at position 1 of the pitch information array. If the absolute value of the difference between the value at the current point and the value at the subsequent point is less then *Thresh* the counter $X$ is increased. If the difference between the value at the current point and the value at the subsequent point is greater than *Thresh* it shows that this part of the pitch information array is not smooth. If the value of $X$ is greater than *Platform*, a long enough segment of the pitch information array was smooth to be considered as a pitch track. The program stores the starting value of this pitch track and now looks ahead into the pitch information array to search for evidence that the pitch track may continue. It will look forward into the array for up to *Flash* points in advance and test if any of the points differs by less *Frange* with the last point of the known pitch track. The value of *Frange* is changed at every data point according to the value of *angle*. If evidence of the continuation of the pitch track is found, the program will once again check *Frange* data points in advance. This process will continue until no further evidence of the pitch track can be found. The location of the identified pitch track is then stored and the program returns to its normal state of scanning through the pitch information array for evidence of a smooth pitch track. When the program has scanned through the complete pitch information array, the process is reversed and the program performs the same operation on the pitch information array, but now starting at the last data point and moving towards the beginning of the array. The information from the forward scan and the backward scan is then combined to give the estimated location of the elephant rumbles.

```
N=Nc
InRumble=1
For c=1 to Nc
Begin
    If A(j)-A(j+c)<=L
    Begin
        N=N-1
    End
End
If  N=0
Begin
    InRumble =0
    P(j…j+Nc)=A(j…j+Nc)
    J=j+Nc
    While InRumble ==0
    Begin
        If  A(j)-A(j+1)<L2
        Begin
            P(j+1)=A(j+1)
        End
            Else
            Begin
                while c<=Nc2 & A(j)-A(j+c)>L2
                Begin
                    c=c+1
                End
            End
        If c<> Nc2
        Begin
            P(j…j+c)=A(j)
        End
        Else
        Begin
            InRumble=1
        End
    End
End
```

Figure 3.37: Pseudo code showing the operation of the pitch track extractor stage of the system.

## 3.6 SUMMARY

This chapter describes the electronic design of an elephant recording collar by first giving the basic requirements of such an device and then proposing a concept design. The details of each of the modules shown in the concept design were discussed. A functional description of the microcontroller software needed to orchestrate the dataflow, file structure and file system was given. The objectives of the mechanical part of the collar were stated and the details of the mechanical design were discussed. In short, the development of a mechanically robust and watertight elephant recording collar designed to record infrasonic sound at a resolution of 16 bits and with a sampling rate of 3 kHz were discussed. The collar has a potential memory capacity of 128 GB onboard which allows for continuous data recording for nine months. The incorporation of a GPS device and a thermometer into the device have also been addressed.

The procedure for the visual detection of elephant rumbles from a spectrogram has been explained. The development of an automatic elephant rumble detection algorithm was discussed. Existing speech processing techniques, namely cochlear filtering and normalized autocorrelations were implemented and combined with a newly devised technique that resulted in an algorithm that can detect elephant rumbles from noisy recordings and also track the pitch of the rumble. A functional description of the implementation of the algorithm in software was given.