DTMF Tone Detection. An Implementation using the TMS320C2xx

Literature Number: BPRA067
Texas Instruments Europe
October 1997

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1997, Texas Instruments Incorporated

Contents

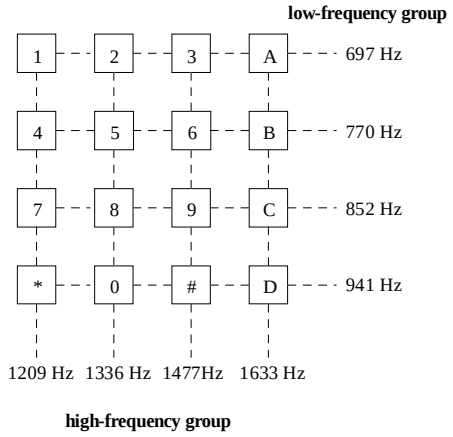
DTMF Tone Detection. An Implementation using the TMS320C2xx

ABSTRACT

This application report deals with the implementation of a dual-tone multiple frequency (DTMF) tone detector on a TMS320C2xx DSP. It describes, in detail, the algorithm and the way of using the routines which have been written for a multi-channel environment. Users can also find information on the process performance and its speed and memory requirements.

1. Introduction

A DTMF transmitter (encoder) generates a composite audio tone burst which comprises two frequencies f_l and f_h that are not harmonically related. Furthermore, the choice of DTMF frequencies has been dictated by the need to avoid confusion between a dialling tone and speech. The correct choice of frequencies makes it highly improbable that any of the possible combinations of frequencies and their levels will be encountered in speech or background noise. The telephone keypad may be used to illustrate the sixteen possible DTMF frequency combinations representing sixteen separate digits.



The signal generated by a DTMF encoder is a direct algebraic summation, in real-time of the amplitudes of two sine (or cosine) waves of different frequencies. $x(t) = A_l \cdot \sin(2\pi f_l t) + A_h \cdot \sin(2\pi f_h t)$.

2. Q.23 Recommendation

The **CEPT** (Conférence des Postes et Télécommunications) has defined the requirements for a central office DTMF receiver to ensure reliable operations. For example, the receiver must tolerate slight variations (frequency bandwidths) in the eight frequencies and the relative signal amplitudes (twist) of the two frequencies comprising a

valid digit. Also, the tone bursts must meet certain timing criteria such as on-off duration etc. The receiver must also reject the dial tone and operate in the presence of certain noise levels. Speech and room noise should not cause the receiver to incorrectly decode or register tone pairs.

2.1 Coding

Signalling frequencies are chosen in two groups of distinct frequencies in the range from 300 Hz to 3400 Hz. A signal is composed of a pair of frequencies - one, and only one, of each group - which are simultaneously transmitted on the line.

Low Group	High Group
697 Hz	1209 Hz
770 Hz	1336 Hz
852 Hz	1477 Hz
941 Hz	1633 Hz

Clearly, none of these frequencies are harmonically related.

The tones are assigned as follows:

Keypad	Low frequency (Hz)	High frequency (Hz)
0	941	1336
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
*	941	1209
#	941	1477
A	697	1633
B	770	1633
C	852	1633
D	941	1633

A typical DTMF receiver uses a special-purpose decoder chip (or chip set) to perform the decode function. Besides performing the main task of DTMF decoding, the TMS320C2xx can also perform a host of other telecom functions, such as A- or μ-law companding (de)modulation etc.

2.2 DTMF Transmitter

Transmitted frequencies are in the range of ± 1.5 % of their nominal value.

Transmission levels are for a resistance of 600 Ω.

1st option -9 dBm ± 2 dB for the high tone

-11dBm ± 2 dB for the low tone

2nd option -6 dBm ± 2 dB for the high tone

-8 dBm ± 2 dB for the low tone

The high-frequency has to be 2 ± 1dB above the low one. This 'offset' allows for the attenuation undergone by the high frequencies in comparison with the low-tones on the telephone transmission lines.

Parasitic signals are specified as being 20 dB less energetic than the low-tone.

Additionally, noise limits are specified according to their position in the frequency spectrum. In particular, for the band from 300 Hz to 4300 Hz, noise must be less than -33 dBm.

A signal is transmitted as long as its corresponding button is pressed.

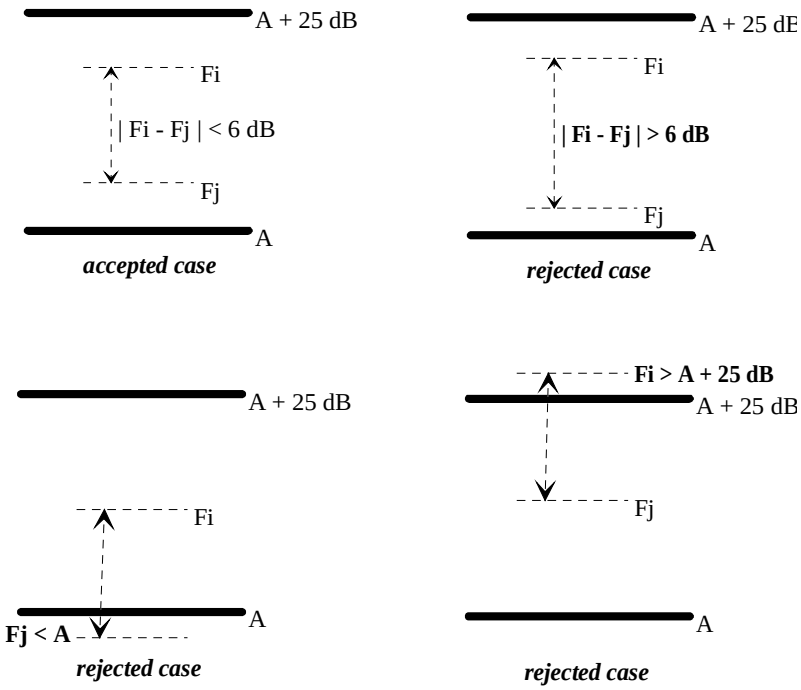
2.3 DTMF Receiver

2.3.1 Frequencies

The signal to be detected comprises two frequencies: one in the low-group and the other in the high group. A tolerance of ± (1.5% + 2Hz) is required.

2.3.2 Power

Their levels are set within limits defined by [A ; A+25 dBm] where A is specified by each telecommunications authority. The level difference is less than 6 dB. In the band [400 Hz : 3400 Hz] the detector must tolerate a noise of 20 dB less than the one of low-group frequencies or an absolute level of (A-22) dB. A twist of 6dB must also be tolerated.

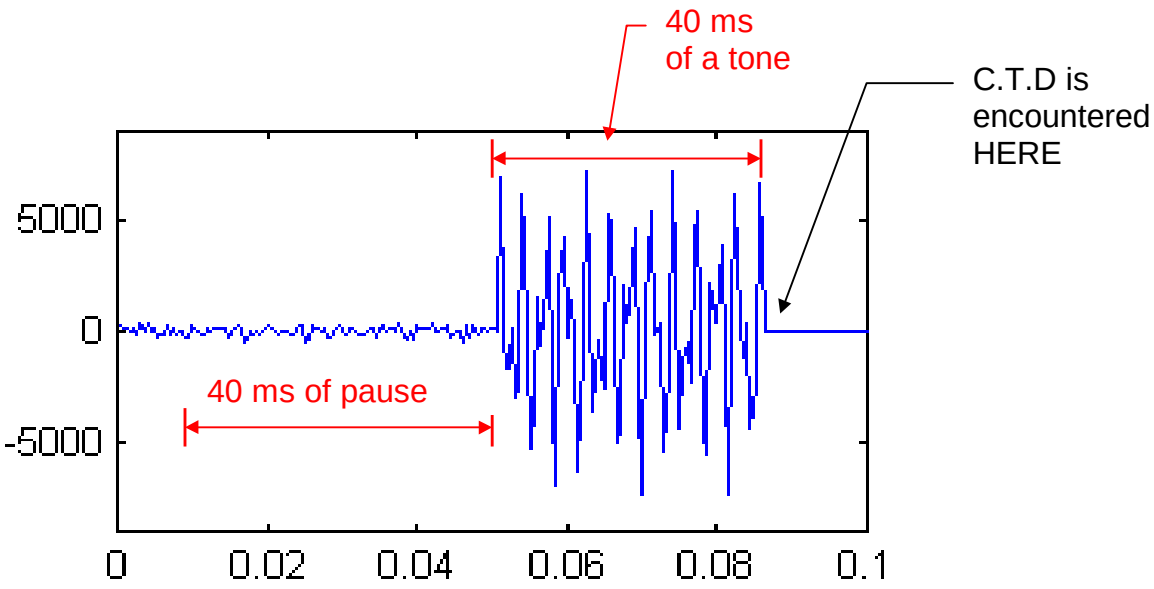


2.3.3 Timing

The detector will recognize a tone if:

- the tone combination is preceded by more than 40 ms of silence (pause),
- this combination is **continuously** present for more than **40 ms.**

These conditions are called, in that report, the “Conditions of Tone Detection” (C.T.D).

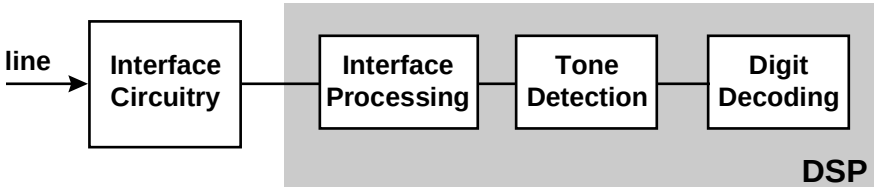

40 ms
of a tone
C.T.D is
encountered
HERE
5000
0
-5000
40 ms of pause
0
0.02
0.04
0.06
0.08
0.1

3. Detector Overview

3.1 Introduction

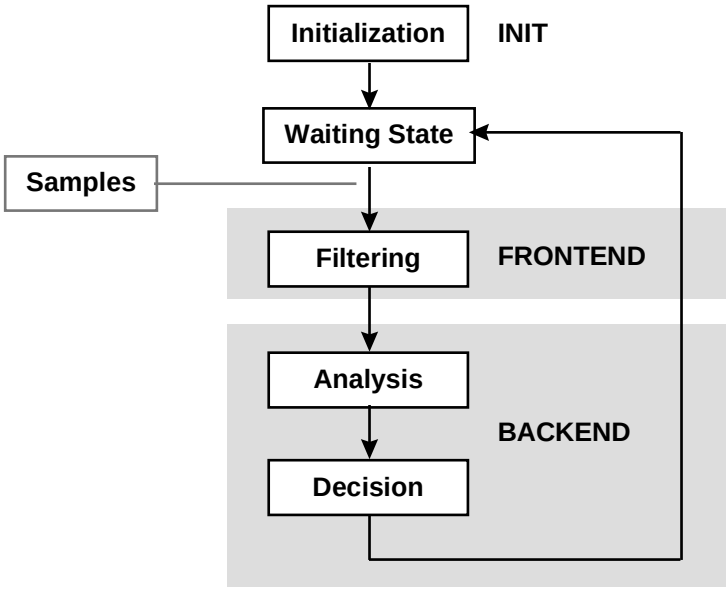
The DTMF detector functionality can be split in two major blocks. The first step deals with tone detection which simply comprises a power detector tuned to the DTMF frequencies. The other stage concerns the specific digit decoding procedures, which, because of the need to safeguard against false digit detection, are relatively complex.

The two tasks differ considerably in their degree of complexity, with the detection process requiring the greater amount of computation. Those considerations lead to the following detector organization.



The part which involves the DSP consists of three major blocks:

- the initialization section (INIT)
- the power detection section (FRONTEND)
- the validation section (BACKEND)



Obviously, FRONTEND and BACKEND have to be run faster than the time between two data inputs. For this reason, a waiting-state concludes each processing cycle. The

filtering block has to be checked on each data input. Furthermore, for the system to work correctly it is essential to ensure that the evaluation is possible in real-time.

At the beginning of a channel assessment, it is necessary to be sure that each node — associated with this channel — of each filter is set to zero and that the energy estimators are empty. After these precautions, the process can start.

3.2 Interface subsystem

As its name implies, this subsystem is in charge of the interface between the signal processor and the outside. Then, by means of software, it formats data into a convenient presentation for processing.

The EVMC5x used to develop this application, is designed for ease of interface. It includes a TLS320C46 AIC which carries out the analog to digital conversion of data. Through this AIC, we can fix the sampling rate of the data flow.

The other role of this interface concerns data conversion. Telecommunication network standards require data transmission in a compressed format (A-law or μ-law). So, before data processing, it is translated from the logarithmic to the linear format. This processing may be simply performed by a look-up table procedure. A 256*B-bit ROM is sufficient for decoding an 8-bits μ-255 coded word into linear coded one which can be subsequently processed by the FRONTEND section.

3.3 Filtering section (FRONTEND)

The FRONTEND section receives appropriately scaled data samples and handles two tasks:

- selective frequency amplification
- power measurement

These outputs feed the next section where results are compared to the norm requirements in order to verify their validity. In fact, both tasks can be treated as one, namely, the evaluation of each DTMF frequency.

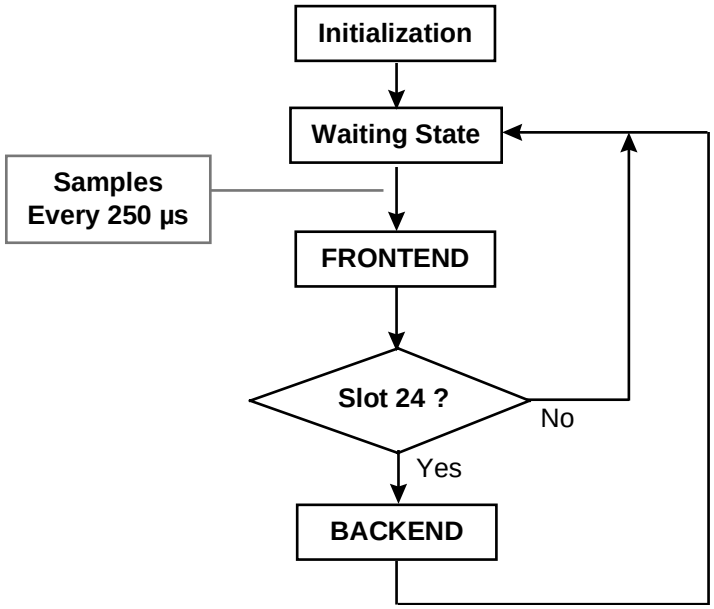
The FRONTEND process is executed at **every sample**. This yields an effective sample rate of 4 kHz.

3.4 Power analysis section (BACKEND)

The receiver backend must track the energy of the receiver's filter outputs. The signal level must be steady within a certain time tolerance in order to be valid. This time is 40 ms. Whereas the data have to be checked at the sampling rate (and so the FRONTEND process has to be run for each new set of data), the BACKEND can be executed less frequently.

3.5 Conclusion

Hence, the general flow chart may be redesigned as follows :



It was decided to execute the **BACKEND every 6 ms,** which is equal to 24 samples of 250 µs each. The allows the BACKEND to be processed every 6 ms for one channel, thus allowing time for additional channel evaluations to be interposed later.

4. Algorithm Choice

In our specific application, it is necessary to:

- *1. identify the frequencies which carry information*
- *2. measure the power of the input signal.*

For this reason we shall look at different methods of power evaluation.

4.1 Power determination

4.1.1 Signal form

As a tone is composed by the addition of two sine waves at two distinct frequencies f_L and f_H, it can be represented in the frequency domain by two Dirac functions, one at f_L and the other at f_H.

$$x(t) = A_L \sin(2\pi F_L t + \varphi_L) + A_H \sin(2\pi F_H t + \varphi_H)$$

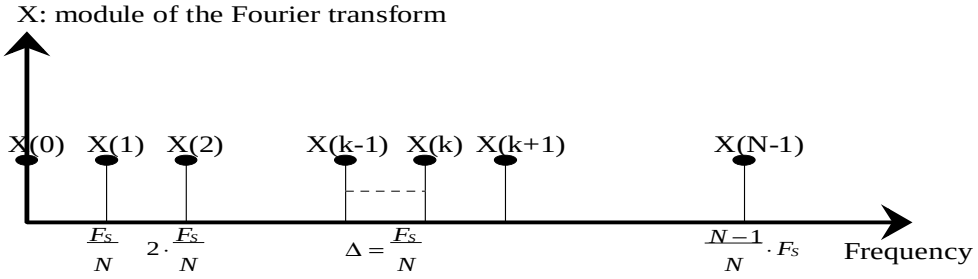
4.1.2 Spectral analysis

4.1.2.1 Fast Fourier Transform

We can consider the DTMF code detection by performing a spectral analysis of the signal by means of a **Digital Fourier Transform** (DFT). Indeed, the two highest channels of the spectrum indicate the pair of received frequencies. Furthermore, it yields a direct evaluation of energy.

By such consideration, we can assume that the tone detection can be carried out by a spectral analysis of the signal by a Fourier transform. With N samples of the signal $x(n)$ (with *n=0..N-1*), we can compute:

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-j.2\pi\frac{kn}{N}} \quad \text{for } k=0..N-1$$



$|X(k)|^2$ represents the energy of *x(n)* at the frequency $\frac{k}{N} F_S$

4.1.2.2 Complexity evaluation

The DFT plays an important role in the analysis, the design and the implementation of digital signal processing algorithms and systems. Furthermore, it is particularly involved in all applications dealing with frequency considerations. Its direct expression is :

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-j.\frac{2\pi\, k\, n}{N}} \quad ,k=0,..,N-1$$

By introducing a factor $W_N = e^{-j.\frac{2\pi}{N}}$ referred to as the **phase factor**, the DFT becomes with $k = 0,..,N-1$:

$$X(k) = \sum_{n=0}^{N-1} x(n).W_N^{kn} \ .$$

The DFT is a complex function and so it may be written as a summation of its real part and its imaginary part :

$$X(k)=\sum_{n=0}^{N-1}\left\{\left(\Re[x(n)].\Re\left[W_N^{kn}\right]-\Im[x(n)].\Im\left[W_N^{kn}\right]\right)+j\left(\Re[x(n)].\Im\left[W_N^{kn}\right]+\Im[x(n)].\Re\left[W_N^{kn}\right]\right)\right\}$$

$$k=0...N-1$$

Thus, to obtain each of the N values of $X(k)$, $4N$ multiplications and $(4N-2)$ additions are necessary. To compute N values, it requires $N(4N-2)$ real additions and $4N^2$ real multiplications. To reduce the complexity of the DFT algorithm, it is wise to study further the relationship between the phase factors W_N^k.

4.2 Goertzel algorithm - Structure of a Goertzel cell

If we are only interested in the spectrum of a signal at one or a few particular frequency points, we can use a special form of DFT called the Goertzel algorithm. The main advantage of this algorithm is that the coefficients in the equation are fixed for a particular frequency point which makes the calculation much simpler. A Goertzel filter is a **very high-Q bandpass filter**, thus characterized by a very selective band. The most common configuration for using this technique is to measure the signal energy before and after the filter and compare both. If the energies are similar then the input signal is centered in the passband, otherwise it is outside the passband. The Goertzel is commonly implemented as a second order IIR filter as shown below.

This method exploits the properties of the coefficients W_n^k. Indeed, since $W_N^{-kN}=e^{-j\frac{2\pi N}{N}}=1$, it is clear that :

$$X(k)=\sum_{n=0}^{N-1}x(n).W_N^{kn}=W_N^{kN}\sum_{n=0}^{N-1}x(n).W_N^{kn}=\sum_{n=0}^{N-1}x(n).W_N^{-k(N-n)}\ .$$

This last expression recalls a convolution:

$$y_k(n)=\sum_{r=0}^{n}x(r).W_N^{-k(n-r)}$$

So we can express $X(k)$ as a function of this convolution:

$$X(k)=y_k(N).W_N^{-k}$$

Observing the periodicity of the phase factors, we find by recurrence:

$$y_k(n)=y_k(n-1).W_N^{-k}+x(n)$$

which corresponds to the difference equation of a digital filter (IIR) with:

$$H(z)=\frac{1}{1-W_N^{-k}.z^{-1}}$$

as system function. We notice that it involves a resonator with a pole on the unit circle at

$$\omega_k=\frac{2\pi k}{N}.$$

To avoid a complex multiply, the above filter $H(z)$ can be multiplied by the complex conjugate pole divided by itself to make the denominator real. This produces the filter:

$$H(z) = \frac{1 - W_N^k . z^{-1}}{1 - 2\cos\left(\frac{2\pi k}{N}\right).z^{-1} + z^{-2}}$$

This filter will evaluate the DFT at the frequency corresponding to the sequence member k for:

$$X(k) = y_k(n)\,|_{n=N}$$

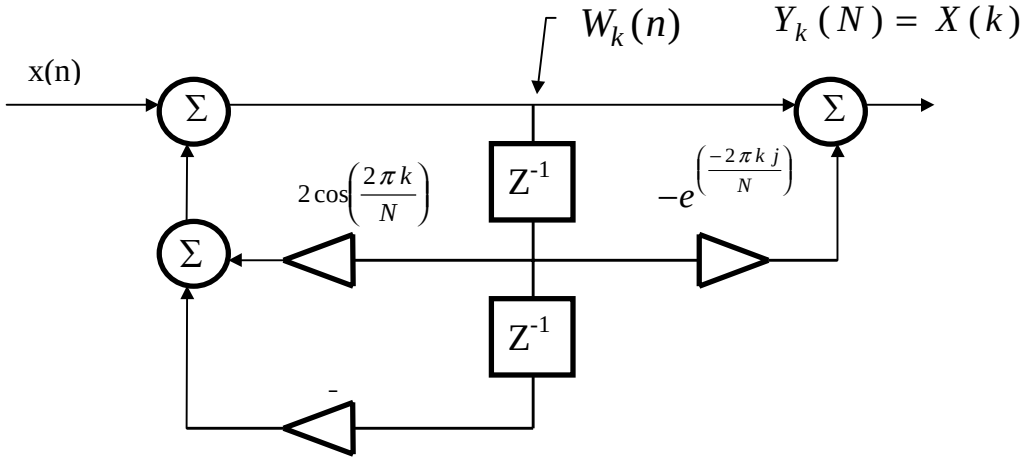
For implementation into a IIR filter, let:

$$\mathrm{H(z)} = \frac{Y(z)}{W(z)} \times \frac{W(z)}{X(z)}$$

which leads to the difference equations:

- $w_k(n) = 2\cos\left(\frac{2\pi k}{N}\right) w_{k-1}(n-1) - w_k(n-2) + x(n)$
- $y_k(n) = w_k(n) - e^{\frac{-j2\pi k}{N}} w_k(n-1)$

which is illustrated in the next figure:



The filter output is only required when $n=N$. The value $w_k(n)$ must be computed N times. If only the power magnitude is necessary the complex multiplication is not required to produce the feed-forward section since:

$$\left|y_k(N)\right|^2 = w_k^2(N) + w_k^2(N-1) - 2\cos\left(\frac{2\pi k}{N}\right) w_k(N) w_k(N-1)$$

4.3 Digital filtering

This principle consists of performing a time filtration. To achieve this, the input signal is applied to as many selective filters -like resonators- as there are different frequencies to

detect. But it only gives one output: the recognized pair of frequencies. Afterwards, the power of filter signals have to be computed or estimated.

The power of a digital signal is expressed by:

$$P(N) = \frac{1}{N} \cdot \sum_{n=1}^{N} \left| x(n) \right|^2$$

where is N is the number of samples.

In fact, a DTMF receiver only deals with power of sinusoids. In fact, the power of the whole signal cannot be interpreted whereas power of each critical frequency can provide the required information. For this reason it is a good idea to analyse the filter outputs. On every filter output in each of the two bands, there is only one sine wave to detect. For example, in lowband, high frequencies are reduced such that their energy is not detectable in this band.

Therefore digital filters can be used to implement a tone receiver in a variety of ways. They select frequencies which are represented in the signal. The main problem is to interpret filter outputs. Obviously, it would not be convenient to compute power by an FFT.

4.3.1 Table look-up method

For a large class of applications, a certain measure of information extraction can be brought about by low-pass filtering. In its simplest form, this can be achieved by a simple averaging of the output data from the digital filter. Such a scheme has been adopted for this DTMF detector.

Basically, multi-frequency receivers, as well as most pieces of transmission test equipment, perform some form of power measurement. Unfortunately, it requires integration of squared data samples, which is an expensive task in term of computation time. The problem is how to implement the squaring. To avoid expensive codes, use can be made of table-lookup. Input samples are used to address the look-up table. The ensuing words, which are proportional to the square of the input samples, are then accumulated serially over a slice (a period to be defined).

Such a power implementation is too expensive for the filter outputs because, as we shall see, the filter output words are too wide (16 bits) and the dynamic range covered is too large to enable use of table look-up.

4.3.2 Power estimation

In the particular case of the DTMF detector, the filters used to split the input signal are narrow-band bandpass filters. Consequently, an input sinusoidal signal is not too much deformed through the bank of filters. This last feature lets us assume that the power can

be **estimated** thanks to the simple integration of the absolute values of the filters outputs. Hence, the power of the f_R frequency-sine over a T-length time is:

$$P = \frac{1}{T} \cdot \int_0^T A^2 \cdot \sin^2(2\pi f_R t) \partial t$$

This function is T_R-periodic. By considering a integration interval T wider than T_R, the average over the T interval is the same as that over a single T_R interval. It becomes:

$$P = \frac{1}{T_t} \cdot \int_0^{T_R} A^2 \cdot \left(\sin\left(2\pi \frac{t}{T_R} \right) \right)^2 \cdot \partial t = \frac{A^2}{2 \cdot T_R} \cdot \int_0^{T_R} \cdot \left[1 - \cos\left(4\pi \frac{t}{T_R} \right) \right] \partial t$$

$$P = \frac{A^2}{2}$$

its associated estimation is:

$$S = \frac{1}{T} \cdot \int_0^T \left| A \cdot \sin\left(2\pi \frac{t}{T} \right) \right| \cdot dt = \frac{2A}{T_R} \int_0^{T_R/2} \sin(2\pi \frac{t}{T_R}) \cdot \partial t = \frac{2}{\pi} \cdot A$$

$$S = \frac{2\sqrt{2}}{\pi} \cdot \sqrt{P}$$

In fact, to avoid a division of the estimator by T the computed estimation is:

$$S = \int_0^T A \cdot \sin(2\pi f_R t) \cdot \partial t = \frac{2\sqrt{2}T}{\pi} \cdot \sqrt{P}$$

where T represents the slice lasting.

For digital applications, power estimation can be:

$$\boxed{S(N) = \sum_{n=1}^{N} |x(n)|}$$

where is N is the number of samples.

4.4 Choice

4.4.1 Advantages of the Goertzel algorithm

This method allows a treatment of each datum as it arrives. By contrast, in the direct DFT algorithm an N-length set has to be recorded before computing (the coefficient pondering $x(n)$ changes at each iteration, so there is a delay of N samples between the reception of $x(n)$ and the determination of $y(n)$).

The advantage of the Goertzel algorithm over the direct evaluation of the DFT described in the previous section is in the reduced number of multiplications (approximately half) and the elimination of memory for storage of node variables. Furthermore, for a DTMF

detection, only eight frequencies matter. Consequently, only eight Goertzel filters have to be implemented.

Node computations — *w(n) = 2*Cst(Scaling)*w(n-1) - w(n-2) + x(n)* — are easier than the resonator ones.

It provides identification of the frequencies and the evaluation of their energies.

4.4.2 Disadvantages of this method

Once N is set, one will have to find the value of k which would tune the Goertzel cell to the desired frequency:

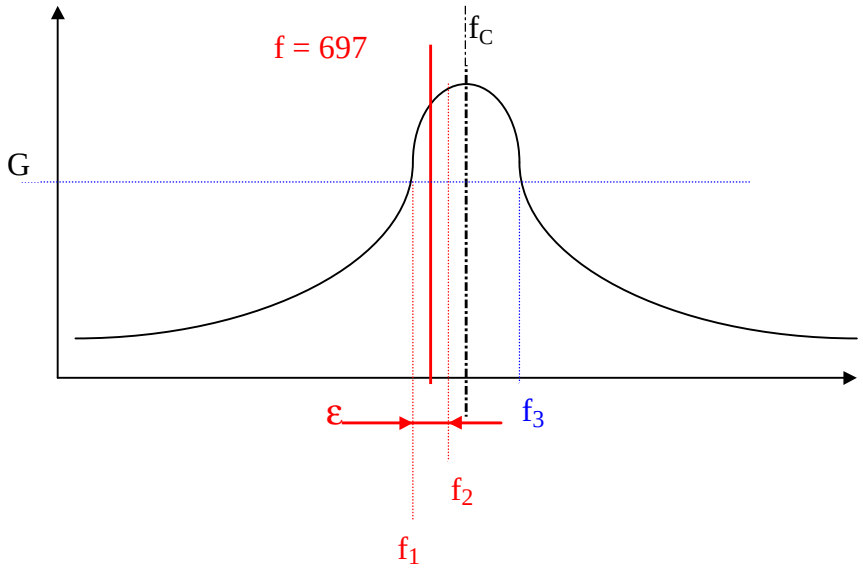
$$\frac{k \times f_s}{N} = f_{desired} \Rightarrow k = \frac{f_{desired} \times N}{f_s}$$

where f_s is the sampling frequency.

The problem lies in the fact that one would have to find a single N for eight cells, and find the eight corresponding *'k's*. It is impossible to find this N ($N \leq 160$ $160 \times 250\mu s = 40ms$) and eight *'k's* which will make cells very well tuned: this method therefore constrains users to tune cells only approximately to the DTMF frequencies.

What's happen in a DTMF detection application if a cell is not well tuned?

The figure below shows the response of such a Goertzel cell (for a 697Hz component detection), when inputs are sine functions.



The application will set a gain G. If the signal is amplified by more than G by the cell, it will mean that the signal includes a 697 Hz component. G would be set according to an ε

admitted error on the received frequencies: so sine waves from the $[f_1;f_2]$ frequency range would be detected. The graph shows us that the cell will force the process to accept undesired frequencies (from the range $]f_2;f_3]$): this is unacceptable.

As appendix B shows, the Goertzel cell can be very sensitive to overflow and roundoff noise because the pole of the system is on the unit circle. The **high gain** of such cells forces the input to be scaled down to avoid overflow. (see appendix B also for more information on *scaling methods*).

Below, a little table compares gains of the Goertzel cell and a resonator, both tuned on 697 Hz.

- G_{lin} is the linear gain
- $\sum$ is $\sum_{m=0}^{N-1}|h(m)|$ where h is the impulse response of the filter nodes.
- Sampling frequency is 4000Hz

	Resonator (ND)		Goertzel
	y node	w node	w node
G_{lin}	48	27	2249
$\sum$, with N= 4000	61	34	2866
$\sum$, with N= 160 (40ms)	59	33	115

As a result, Goertzel cells must be more heavily scaled down than resonators. This table also stresses properties of the Goertzel cells: they are on the limit of stability.

4.5 Conclusion

An IIR filter followed by a power estimator makes it possible to obtain an acceptable estimation of the signal power at EACH SAMPLE. This kind of filter results from a *time process* whereas the Goertzel algorithm acts as *a frequency-based treatment by frame.*

A Goertzel cell is on the limit of stability (its impulse response is a sine function) while resonators can be built to be **stable**.

Unlike resonators, Goertzel cells are not very well tuned on the DTMF frequencies. Furthermore, resonators can be made to optimize overflow management and roundoff effect.

As a consequence, we have selected the filtering method.

5. Data Format

5.1 Number representation

The fixed point mode consists of fixing the position of the binary comma in the binary word. The first bit of the word is dedicated to the sign expression. The addition of two fixed-point numbers on N bits is not erroneous but it requires an $(N+1)^{th}$ bit. On the other hand, the multiplication of such words involves a rounding or a truncation error.

- The sign-magnitude representation divides the word into two fields, the sign field on one bit and the absolute value field on the (N-1) other bits.
  For instance, on 5 bits 0.375 becomes 0.0110
  -0.375 becomes 1.0110

Thus, an N-bit word can take 2^N-1 different values. The zero can be represented in two different ways (0.0000 or 1.0000 on 5 bits). Furthermore, the sign-magnitude method is not practical to apply for computations (particularly for additions).

- The individual inversion of all bits in the word can provide a number inversion. It is known as the 1's-complement method. Here, on N bits, 2^N-1 values are free. Keeping the last example, we have :

  0.0110 for 0.375 and
  1.1001 for -0.375

- In the preceding systems, one value is lost because of the double representation of zero (0.0000 and 1.1111 on 5 bits). To avoid this, another representation is generally used : the 2's complement method. To change the sign of a number, all bits are inverted and a 1 is added. And, so, 2^N combinations of an N-bit word become viable.

  For example : 0.375 is given by 0.0110 and
  -0.375 is given by 1.1010

5.2 Data coming from the A/D converter (AIC)

The AIC, which carries out the A/D and D/A conversions, receives input voltages between -3V and +3V. It quantifies those data on 16 bits at an 8kHz sampling rate. Therefore, the 2 LSB are reserved for dialogue between the DSP and the AIC.

$\begin{cases} x_q : \text{quantified data on a 16 bits length word with the 2's complement method.} \\ x_{an} : \text{corresponding analog input/output} \end{cases}$

$x_q = \frac{x_{an}}{V_{max}} \times D_{max}$ with, according to EVM feature, $\begin{cases} D_{max} = 32767 = 2^{15}\text{-}1 \\ V_{max} = 3V > x_{an} \end{cases}$

The MSB contains sign information. That format is usually called Q.15 format. Here is its representation:

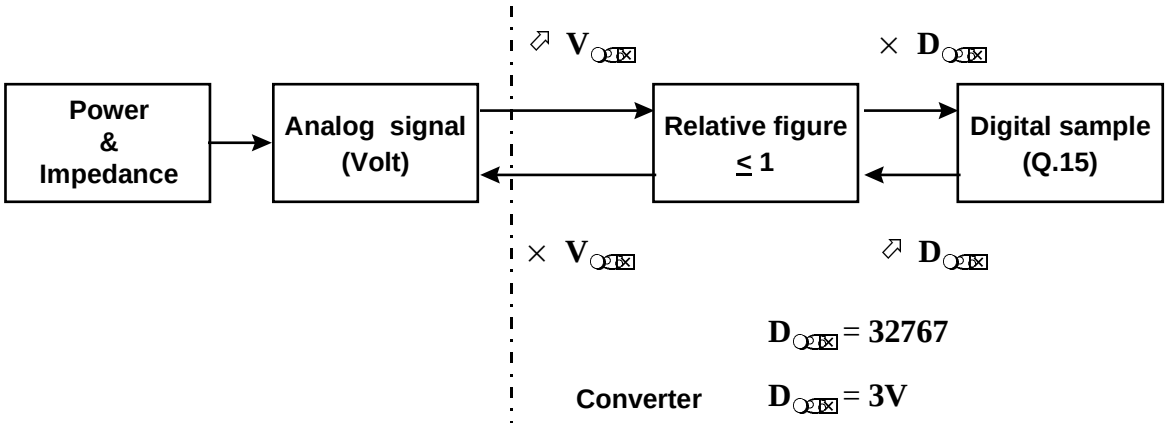
$x_{an} / V_{max} =$

S	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

$x_q =$

S	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.3 Conclusion



If Vmax from your converter is different, you will have to modify - by a multiply inside the routine, for example - values of samples or decision parameter values (MINTH, MINTL ...).

WARNING !!

The AIC (TLC32046) sampling frequency has been set to 8kHz but the algorithm needs a 4kHz sampling rate (16bit×4kHz = 64kbit/s), so 1 sample over 2 must be treated by the routine.

6. Detector description

6.1 Filtering section

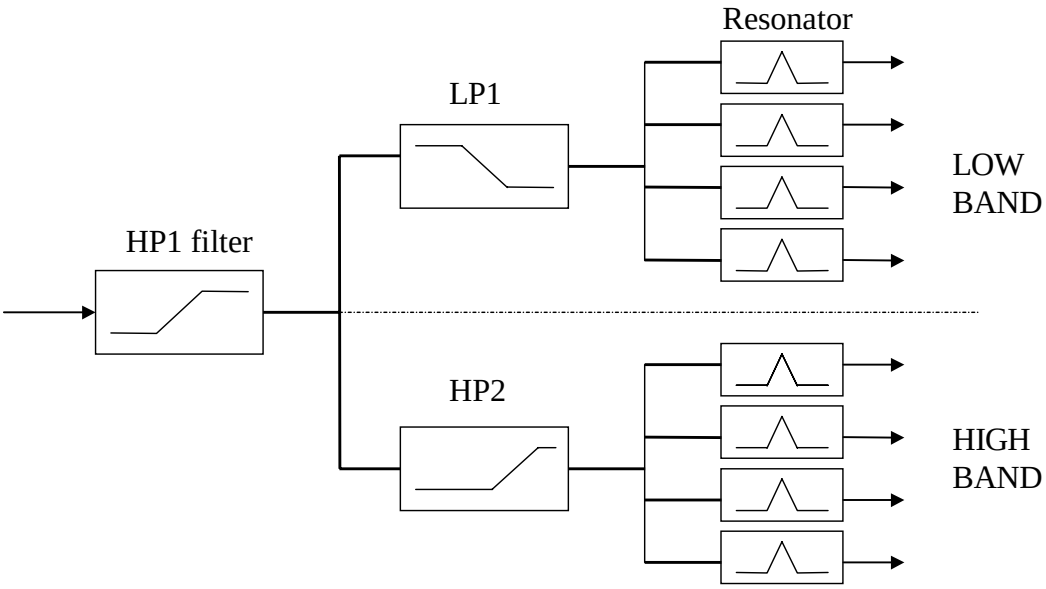
This section is also called ‘FRONTEND’ section in this document.

6.1.1 Structure choice

The CEPT lays down that the detector should tolerate parasitic frequencies. These unwanted noise signals have to be filtered, particularly the 50 Hz component, which remains very energetic till its third or fourth harmonic. Furthermore, as we have seen earlier in this presentation, the filtering process has to decide in less than 40 ms. This feature limits the tightness of the filters and a division of the band in two sub-bands reduces the number of necessary coefficients. The signal is not cut in highband because we have considered the ideal case of a pure tone without any noise, aliasing or other undesirable effects.

Taking into account the acceptable frequency variation, we can distinguish two bands: the low band from 684 Hz to 957 Hz and the high band from 1188 Hz to 1660 Hz.

Here is a scheme of the filtering structure:



The first filter HP1 eliminates very low frequencies below 680 Hz.

The LP1 and HP2 cells which separate bands must, at least, enclose the bandwidths [680 ; 960] and [1180 ; 1670]. Moreover, to reduce as much as possible the order of filters, they are not too high. A resonator amplifies the frequency on which it is tuned. So resonators must be centred on the frequencies 697, 770, 852, 941, 1209, 1336, 1477 and 1633 Hz.

6.1.2 HP1-LP1-HP2 filters

6.1.2.1 Specification of the filters

The sampling frequency is 4 kHz.

	HP1	**LP1**	**HP2**
Fcut (Hz)	480	1230	1020
Fband (Hz)	580	1100	1230
Passband ripple	0.249	0.249	0.249
Stopband ripple	0.249	0.249	0.2

6.1.2.2 Evaluation of filter order

Once those parameters are specified to DFDP, it evaluates the order for each family of filters.

TYPE	Type/Window	LP1	HP1	HP2
IIR	Butterworth	7	7	6
IIR	Chebychev I	3	3	3
IIR	Chebychev II	3	3	3
IIR	**Elliptic**	**2**	**2**	**2**
FIR	Kaiser	12	15	9
FIR	Rectangular	61	81	45
FIR	Triangular	121	159	87
FIR	Hanning	121	159	87
FIR	Hamming	128	161	89
FIR	Blackman	184	241	133
FIR	Remez	10	13	9

6.1.2.3 Determination of coefficients

To limit the processing time as much as possible, the **elliptic** filter solution had been chosen. According to the difference equation:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + a_1 y(n-1) + a_2 y(n-2)$$

their respective coefficients are:

	HP1	LP1	HP2
a_1	0.8845215	-0.0498047	-0.7675781
a_2	-0.6782227	-0.6328125	-0.6206055
b_0	0.5310059	0.4124756	0.2677002
b_1	-0.8710938	0.4433594	-0.1231689
b_2	0.5310059	0.412475756	0.2677002

As required, frequencies under 560 Hz are rejected by HP1. Then LP1 (or HP2 respectively) only keeps frequencies in the range [580 Hz;1125 Hz] (or [1260 Hz;...]). We notice that the high group is not cut off but this is not significant for an ideal case. To limit the number of registers, those filters do not correspond to a direct structure. Furthermore, such bandsplit filters must avoid overvoltage effects; this argument eliminates the DN solution. So, they have been implemented according to the ND structure (see appendix B for more information).

6.1.2.4 Scaling factor

(see annexe B for more information)

- **HP1**

It is preceded by a scaling factor α_0. Using Matlab, we have computed *HP1*$_W$ and *HP1*$_Q$ prototypes which correspond to the transfer functions at both its quantization points.

$$\begin{cases} \underset{\omega}{Max}\left|HP1_W(\omega)\right| = 1.2567 \\ \underset{\omega}{Max}\left|HP1_Q(\omega)\right| = 0.9513 \end{cases}$$

Thus, for an input such as $|x(n)| < 1$ and quantization points such as

$$\omega_{\max} = 1 \text{ and } Q_{\max} = 1$$

$$\alpha_0 = Min\left(\frac{1}{1.2567}; \frac{1}{0.9513}\right) = 0.796$$

which is the strictest constraint.

- **LP1**

We have determined $\alpha_0 = 0.796$. So we are able to estimate α_1 up-side LP1:

$$\alpha_1 \approx Inf\left(\frac{1}{\underset{\omega}{Max}\left|\alpha_0 \cdot HP1(\omega) \cdot LP1_Q(\omega)\right|}; \frac{1}{\underset{\omega}{Max}\left|\alpha_0 \cdot HP1(\omega) \cdot LP1_W(\omega)\right|}\right)$$

By a Matlab simulation, we have determined

$$\alpha_1 = Inf\left(\frac{1}{0.60338}; \frac{1}{0.8938}\right) = 1.119 > 1$$

As it is greater than 1, this scaling factor is ignored.

- **HP2**

We have determined $\alpha_0 = 0.796$. So, we are able to estimate α_2 up-side HP2:

$$\alpha_2 \approx Inf\left(\frac{1}{\underset{\omega}{Max}\left|\alpha_0 \cdot HP1(\omega) \cdot HP2_Q(\omega)\right|}; \frac{1}{\underset{\omega}{Max}\left|\alpha_0 \cdot HP1(\omega) \cdot HP2_W(\omega)\right|}\right).$$

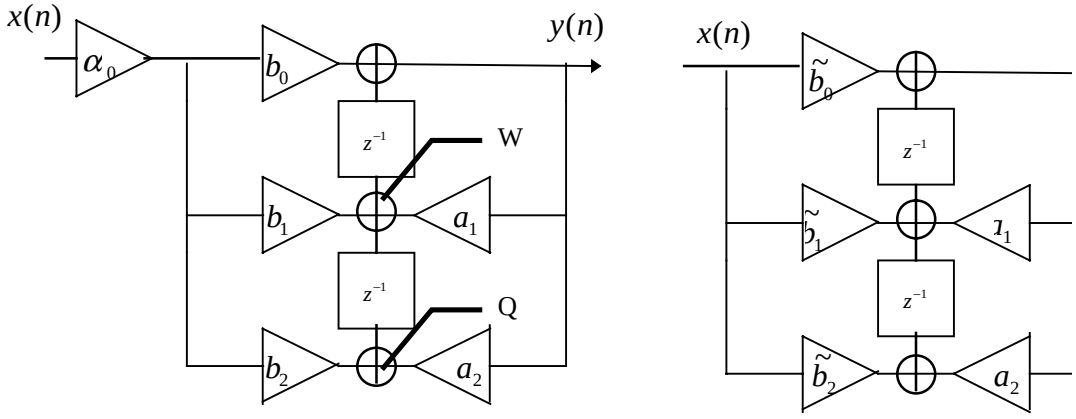
By a Matlab simulation, we have determined:

$$\alpha_2 = Inf\left(\frac{1}{0.4812}; \frac{1}{0.0.7515}\right) = 1.3307 > 1.$$

As it is greater than 1, this scaling factor is also ignored.

- **Implication for filter coefficients**

To reduce the number of operations, let us observe the ND structure more closely.



The scaling factor can be adapted to the numerator coefficients. This ploy saves a multiplication per input sample. The resulting coefficients are:

	HP1	LP1	HP2
a_1	0.8845215	-0.0498047	-0.7675781
a_2	-0.6782227	-0.6328125	-0.6206055
b_0	0.422640773	0.4124756	0.2677002
b_1	-0.693472074	0.4433594	-0.1231689
b_2	0.422640773	0.412475756	0.2677002

6.1.2.5 Sizing

Filtering input comes from the AIC which sized data on 16 bits.

node sizing	S	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
coefficient sizing	S	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	
coefficient x *node*	S	S	2^{14}	2^{13}	2^{12}	...	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-13}	2^{-14}	2^{-15}
coefficient x *node*	S	2^{14}	2^{13}	2^{12}	2^{11}	...	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	...	2^{-14}	2^{-15}	0
other node (16 bit shifted)	S	2^{14}	2^{13}	2^{12}	2^{11}	...	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0	0	0	0	0	...	0	0	0

This is why coefficients are quantified on 16 bits according to the Q15 format: $\alpha_q = \alpha \cdot 2^{15}$.

Resulting set of coefficients

	HP1	LP1	HP2
A_1	28992	-1635	-25152
A_2	-22222	-20733	-20323
B_0	13849	13517	8772
B_1	-22724	14528	-4036

B_2	13849	13517	8772

6.1.3 Resonators

The role of those elements consists of the separation of individual DTMF frequencies. In fact, those we have implemented are not real resonators. Indeed, according to the CEPT requirements a frequency has to be validated as soon as it is contained in the range $F_R \pm (1{,}5\,\% + 2\ Hz)$. Now resonators (particularly Goertzel cells) are very narrow band filters. Consequently, we have implemented narrow band filters large enough to meet the CEPT recommendation in terms of frequency sliding.

The characteristic behaviour of a given filter depends on the placement of its poles and zeroes (poles correspond to input excitations which cancel the z transform denominator and zeros correspond to those which cancel its numerator). They are studied with respect to the unit circle.

Consequently, to evaluate a narrow band filter, we have to consider its zero and pole placements.

The presence of a zero close to the unit circle will cause the magnitude of the frequency response to be small - for frequencies that correspond to points of the unit circle close to that point. In contrast, the presence of a pole close to the unit circle will cause the magnitude of the frequency response to be large at frequencies close to that point. Placing a zero close to a pole cancels the effect of pole and vice-versa.

The basic principles underlying the pole-zero placement method is to locate poles near points of the unit circle corresponding to frequencies to be emphasized and to place zeros near the frequencies to be de-emphasized. Furthermore, all poles should be placed inside the unit circle in order for the filter to be stable. To work with real coefficients, we arrange that all poles and zeros should occur in complex conjugate pairs. Basically a band pass filter contains one (or more) pairs of complex conjugate poles near the unit circle, in the vicinity of the frequency band that constitutes the pass band of the filter.

6.1.3.1 Computation of coefficients

To improve the performance of the resonators, they should conform to the z-transform:

$$H(z) = \frac{1 + Cz^{-1} + Dz^{-2}}{1 - 2Cz^{-1} - Dz^{-2}} \quad \text{with} \quad \begin{cases} C = e^{-aT} \cdot \cos\left(\dfrac{2\pi\, fc}{fs}\right) \\ D = -e^{-aT} \end{cases}$$

e^{-aT} : pole radius
fc : centre frequency
fs : sampling frequency

and the coefficients are :

	C	C quantified	D	D quantified
697	0.454589843	14690	-0.984375	-31366
770	0.350097656	11305	-0.981201171	-31221
852	0.227783203	7350	-0.978271484	-31061
941	0.091308593	2943	-0.975097656	-30887
1209	-0.316984531	-10171	-0.966064453	-30369
1336	-0.493408203	-15824	-0.959960937	-30130
1477	-0.665283203	-21306	-0.953857421	-29865
1633	-0.816162109	-26600	-0.947998046	-29573

6.1.3.2 Scaling factor

To limit time consumption of cycles, there is only one scaling factor before the low resonator bank and another before the high resonator bank.

With Matlab, the four highest gains of **the filter cascade HPI-LPI-Resonator** of the low/high group were computed. For each resonator it provided four values so we just have to select the highest one to know the corresponding scaling factor.

Chain filters extremes:

Resl1	Resl2	Resl3	Resl4	Resh1	Resh2	Resh3	Resh4
24,8184	21,9134	19,0358	16,8229	6,5814	10,8991	10,7180	13,2157

It results two scaling factors:

$$\alpha_L = 0{,}0403$$

$$\alpha_H = 0{,}0757$$

In order to avoid another multiplication on LPI or HP2 outputs, those scalings are carried out by a shift of resonator inputs - since this involves no additional time.

$$\alpha_L \geq 2^{-5} = 0{,}03125$$

$$\alpha_H \geq 2^{-4} = 0{,}0625$$

6.1.4 Power computation - Integration

In fact, this consists only of an accumulation of absolute values of filter outputs. To avoid overflow, they are resized (shifted to the right) before being considered. Data are integrated on a slice (24 slots). We have simulated through Matlab the worst cases (most energetic tones) and we noticed the resulting highest values of estimators LY,LYi, HY and Hyi. Thus it gave us the resulting resizing factors:

on lowband estimators: 2 bits to the right
on highband estimators: 2 bits to the right
on lowband individual estimators: 3 bits to the right

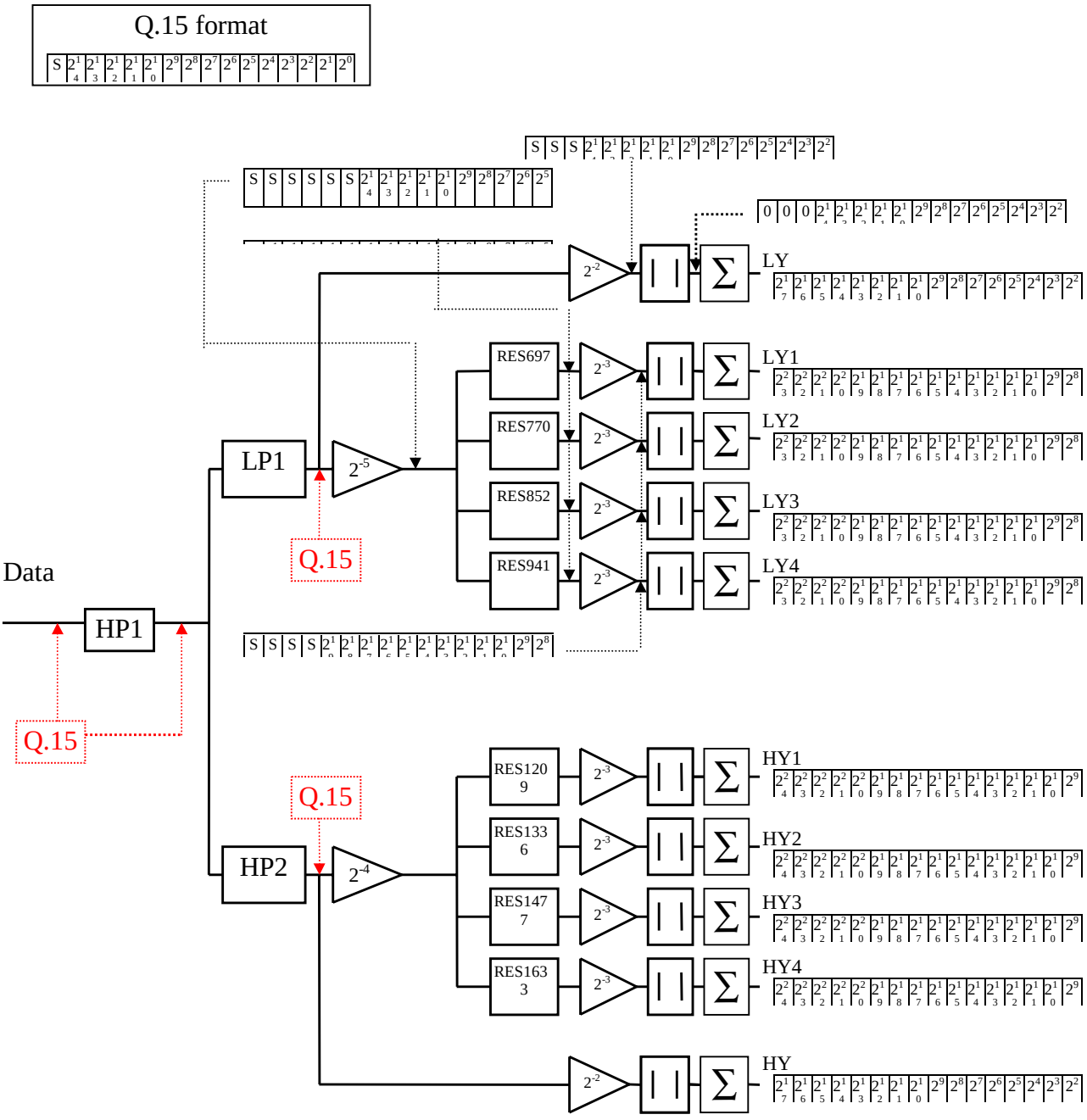
on lowband individual estimators: 3 bits to the right

6.1.5 Results

No overflow can occur while power levels are as described in Q.23 recommendations and while the A/D converter is similar to the one described in ‘DATA FORMAT’ chapter. To protect the process against very highly energetic signals, the DSP will run in overflow mode (when an overflow occurs, the overflow flag is set, and the accumulator is loaded with either the most positive or the most negative value that can be represented in the accumulator depending upon the direction of the overflow).

In the appendices, you can find the prototype of the filters and resonators.

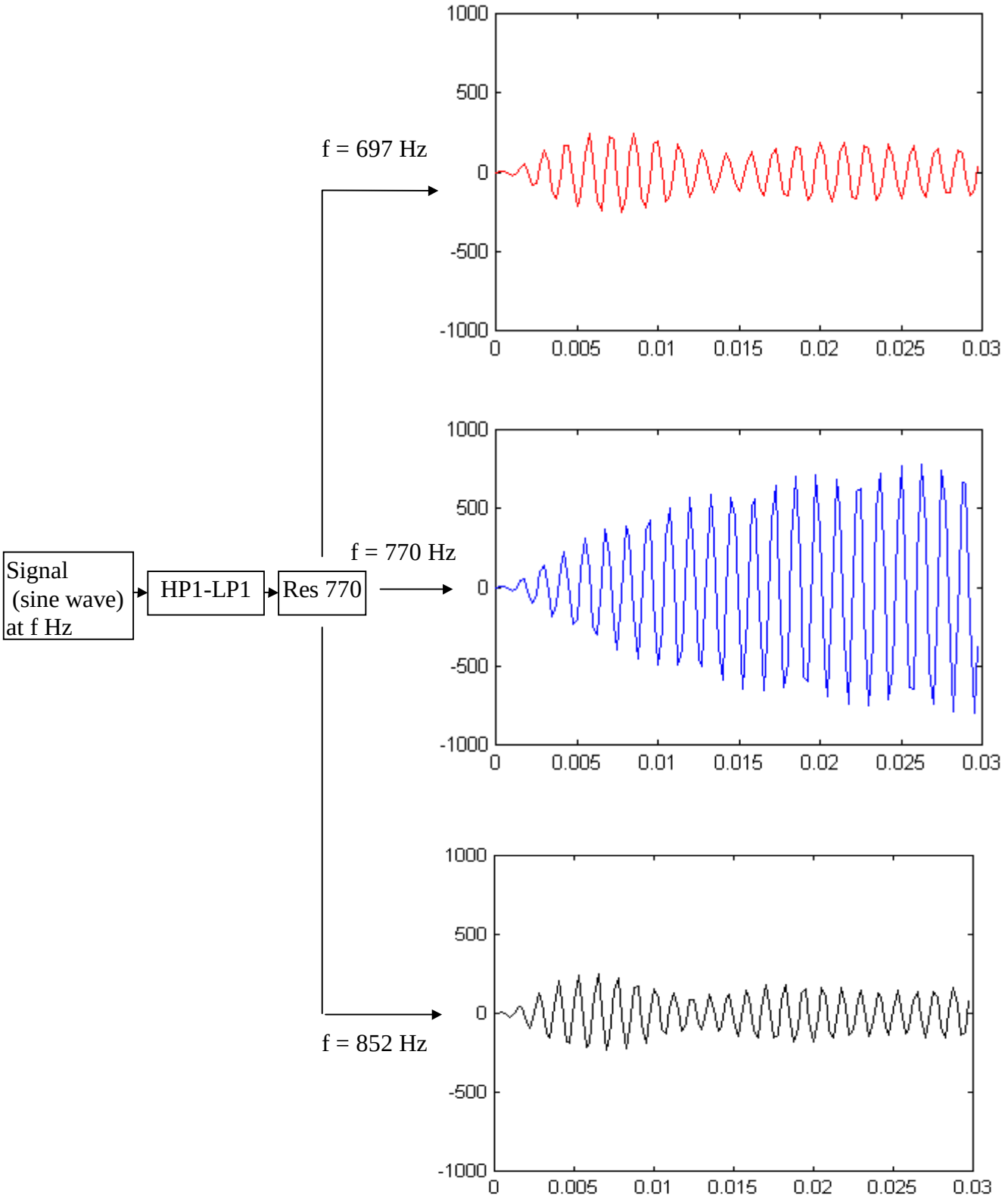
The next scheme will show the filtering structure resulting, according to the data re-sizing.



6.1.5.1 Resonator transitory responses

One should note that a filter, especially a resonator, always needs a certain settling time.

This time depends on the filter structure. Using MATLAB, we have plotted this response (with a sine wave excitation) for the 770 Hz resonator. The resonator would generate a sine wave (at the output) if the input is a sine wave with the *correct* frequency – actually, the frequency it is tuned to.


Signal
(sine wave)
at f Hz
HP1-LP1
Res 770
f = 697 Hz
f = 770 Hz
f = 852 Hz
1000
500
0
-500
-1000
0
0.005
0.01
0.015
0.02
0.025
0.03

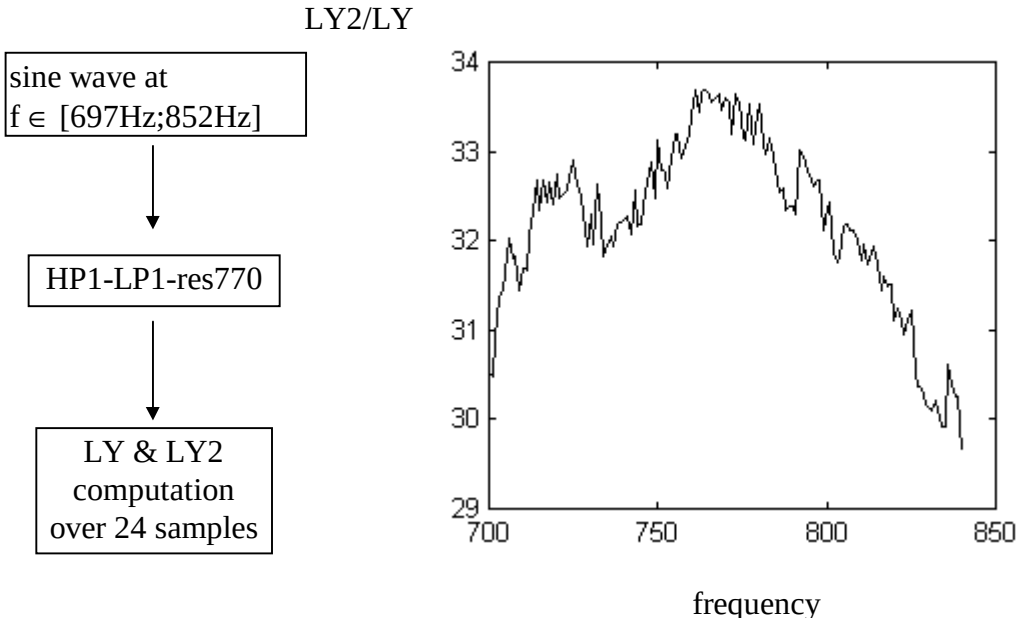
6.1.5.2 Consequences for the analysis

The power analysis will take care of the signal amplification in various resonators. For example, if amplification at the 770 Hz resonator is below a threshold (not even determined), it would mean that a sine wave at about 770 Hz is included in the signal.

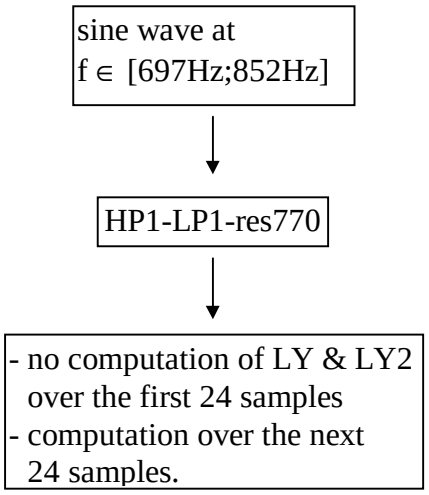
The first 6 ms cannot be well analyzed: outputs of the resonator for different sine waves are about the same during this 6 ms.

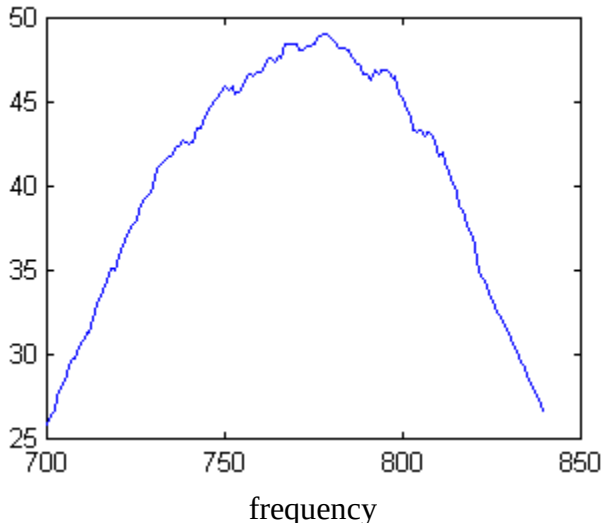
But when can the output of the resonator be analyzed?

The graphs which are shown below can help us to answer that question.

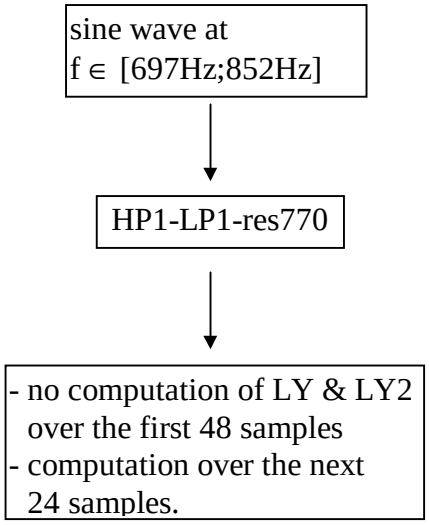


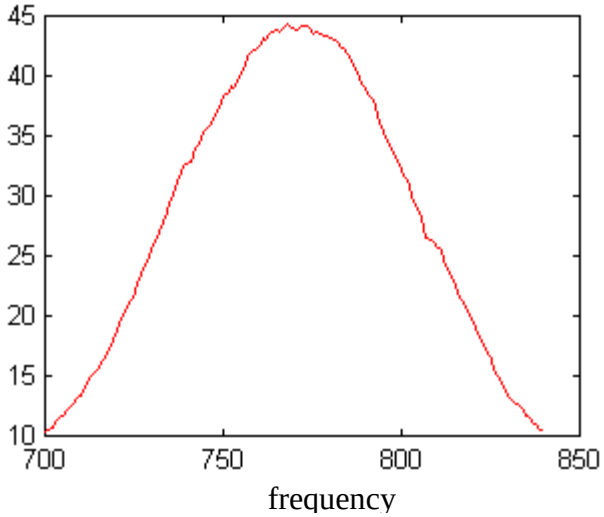
LY2/LY





LY2/LY





- In the first the case: The amplification of the signal (LY2/LY) cannot be interpreted: figures seem to be too noisy. And it is not surprising as it had been predicted by the previous analysis, in time domain.
- In the second case: LY2/LY looks better. Waiting for the next 24 samples and let the first ones down could be the right solution.
- In the third case: LY2/LY is very much better. It does not seem noisy and its graph is narrower than the one from the second case.

In the ideal case, the decision process has to wait for 6 successive identical results before declaring the detection of a tone (6*6 ms ≈ 40 ms). But according to the resonator

transient response, IT IS BETTER TO WAIT FOR 4 SUCCESSIVE IDENTICAL RESULTS before declaring the detection of a tone (6-2=4; 6, from the ideal case - 2, because of the reasons in the *first and second cases above*).

6.2 Power analysis section

This section is also called 'BACKEND' section in this document. It is divided into two main sub-sections:

- the test section
- the decision section

The test section will determine if the received signal is a tone or not (if it is not, the signal is called a pause).

Then the decision section will act and transmit its conclusion according to the test section result.

6.2.1 Test section

All parameter evaluations were done by means of MATLAB and some of them were modified during the practical algorithm validation (see also the 'Algorithm validation' chapter).

6.2.1.1 Test inventory

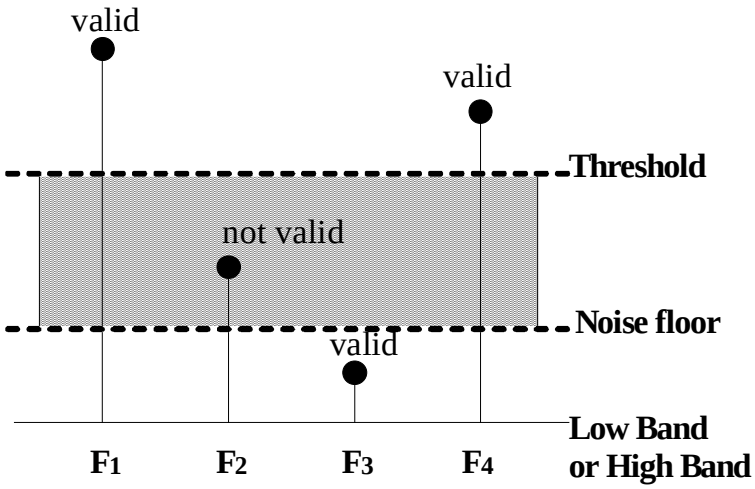
First, the input signal level is checked against the *minimum signal level*. Thus, we ensure that band power is higher than minimum threshold.

Then the *signal swing* is tested. The signal swing is a good measure of whether the input signal is DTMF or speech. If the signal is DTMF it is fairly constant (+/-1 dB); speech, on the other hand, causes a much wider signal swing.

Then the *individual frequencies* present in the signal are determined. This process takes place after a stabilization time of the very narrow digital resonators.

For each frequency the input is checked to see if its normalized energy estimate is sufficiently high to be a possibly valid frequency, or, in contrast, sufficiently low that it can be considered to be noise.

This process can also be described as *centre clipping* with *automatic gain control* (AGC).

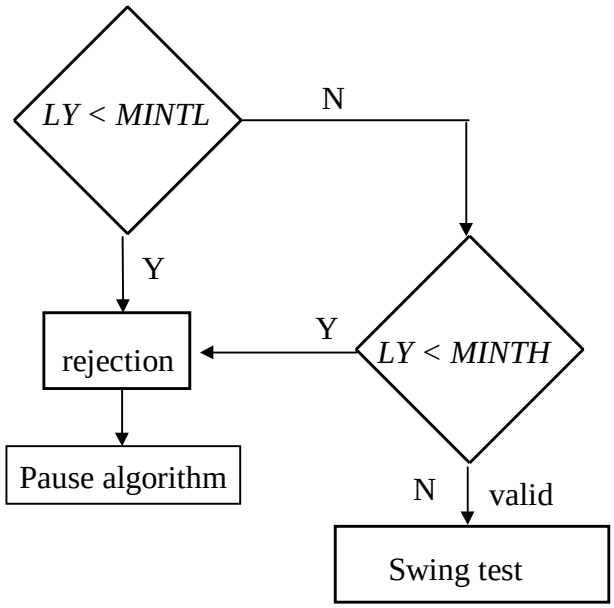


This process is carried out for all 8 frequencies. In order to yield a valid DTMF combination, only one frequency of each band must be above the threshold. All others have to be lower than their noise floor threshold. The valid frequency-pair serves as address offset into the table NUMBER.

If a digit is detected, a *twist test* is performed by comparing the averaged signal energies of the lowband and the highband. Each digit has its own twist coefficients: this had been done because of the ripple level of HP1-LP1 and HP1-HP2 filters.

6.2.1.2 Threshold test

Flowchart: To be sure that the power runs past a minimal value, lowband and highband estimators are compared to a threshold.



Parameter evaluation: For all 16 possible combinations,

- simulate the FRONTEND process over 40 slices
- select the minimum LY and HY values among all those simulations.

We have obtained MINTL and MINTH values:

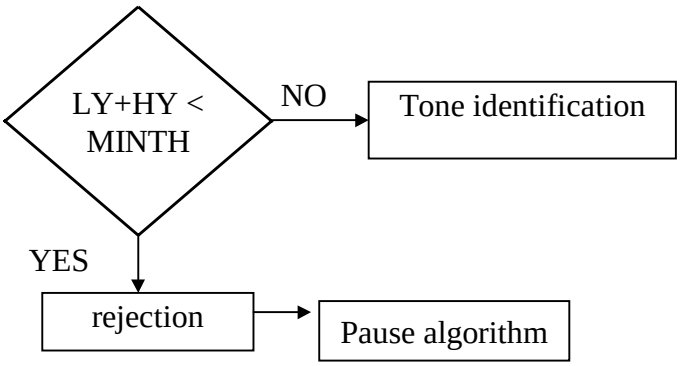
MINTL=2297
MINTH=1099

Those values are directly compared to LY and HY so they are identically sized.

6.2.1.3 Adaptative threshold test

(swing test used in speech immunity)

The DTMF signal level is stable: (LY+HY) is about constant (level swing is ± 1dB). So, HY+LY is compared to the latest (HY+LY)/2 evaluation which was stored into MINTH. LY,HY and MINTH are identically sized.



6.2.1.4 Tone identification - Process

The tone is identified by comparing the gain estimations on resonator outputs to their individual thresholds and to the noise floor. As soon as a frequency is recognized, an index is increased. This variable allows a table to be read at the corresponding address and, thus, provides the right dialled tone.

To validate the f_i frequency, two conditions are required:

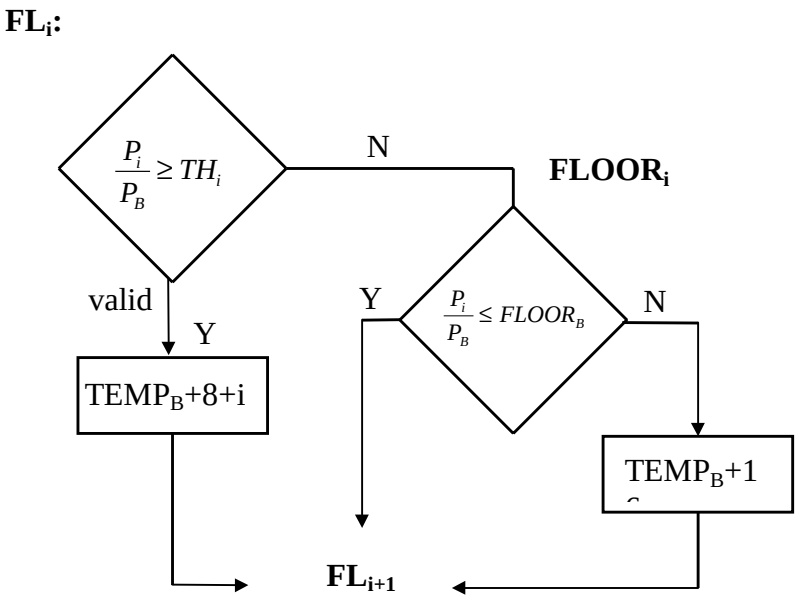
$$\begin{cases} \dfrac{P_i}{P_B} \geq Th_i \\ \dfrac{P_{j\neq i}}{P_B} < FLOORB \end{cases}$$

Th_i is the minimum gain the signal must have if it includes a DTMF frequency. 'Gain' is the result of (LY_i/LY), so it is not the theoretical way of computing it but an equivalent way.

If the signal is a DTMF tone without the i^{th} component, LY_i/LY must be below FLOORB.

Otherwise, the signal gain is between FLOORB and Th_i.

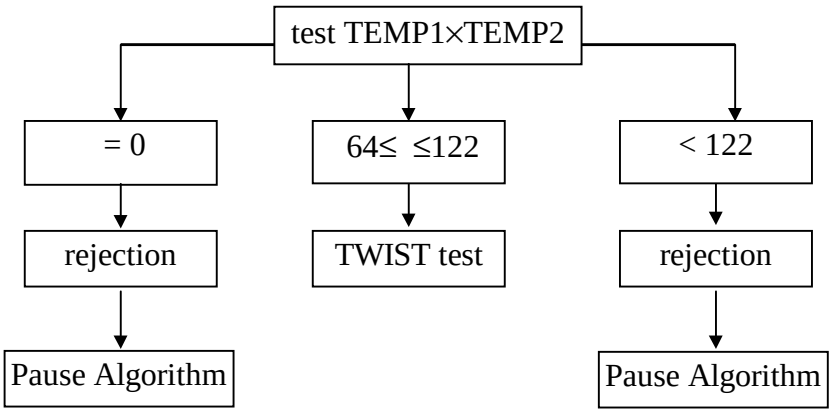
For each frequency, the process below is followed:



P_i: power estimator on the i[th] resonator
P_B: power estimator in the band
Th_i: threshold of the i[th] frequency
$TEMP_B$: index of the band
TEMP1 for low band
TEMP2 for high band

At the end of the process:

if $TEMP_B \geq 16$, there is too much noise, there is no validation
if $TEMP_B = 0$, there is no valid frequency
if $TEMP_B = 8;9;10;11$, validation of f_1, f_2, f_3 or f_4 (respectively)



6.2.1.5 Tone identification - Computation of coefficients

There are ten coefficients to determine:

- the four threshold parameters in low band TLi
- the four threshold parameters in high band THj
- the noise floor parameter in low band MINTL
- the noise floor parameter in high band MINTH

They all have been computed by means of a Matlab simulation. This tests the minimum and maximum values of all the power estimators: *LY, LY1, LY2, LY3, LY4, HY, HY1, HY2, HY3* and *HY4*. Many of them were modified during practical tests.

6.2.1.6 Tone identification - Parameter sizing

The estimators are not identically sized:

LY	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}
LY_i	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}

Obviously, we have to take this into account. That is why a factor acts on the parameters.

low_param are defined as: $\begin{cases} TL_i = \min\left[\dfrac{(LY_i)}{(LY)} \cdot \dfrac{2^{23}}{2^{17}}\right] = 2^6 \cdot \min\left[\dfrac{(LY_i)}{(LY)}\right] \\ FLOORL = \max\left[\dfrac{(LY_{j\neq i})}{(LY)} \cdot \dfrac{2^{23}}{2^{17}}\right] = 2^6 \cdot \max\left[\dfrac{(LY_{j\neq i})}{(LY)}\right] \end{cases}$

Furthermore, this factor is not the same for high band parameters:

HY	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}
HY_i	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}

high_param are defined as: $\begin{cases} TH_i = \min\left[\frac{(HY_i)}{(HY)} \cdot \frac{2^{22}}{2^{17}}\right] = 2^5 \cdot \min\left[\frac{(HY_i)}{(HY)}\right] \\ FLOORH = \max\left[\frac{(HY_{j\neq i})}{(HY)} \cdot \frac{2^{22}}{2^{17}}\right] = 2^5 \cdot \max\left[\frac{(HY_{j\neq i})}{(HY)}\right] \end{cases}$

First, we can try to fit the parameters within 16 bits. Unfortunately, this cannot be done because of sign consideration, even in unsigned multiplication.

Low_Param	0	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
TL_i or FLOORL	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

(This sizing implies a normalization factor of 2^8=256.)

LY.Low_Param	0	0	2^{23}	2^{22}	2^{21}	....	2^{11}	2^{10}	2^{9}	2^{8}	...	2^{-4}	2^{-5}	2^{-6}		
↓ PM=01																
LY.Low_Param	0	2^{23}	2^{22}	2^{21}	....	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	...	2^{-5}	2^{-6}	0		
$LY_i.2^{15}$	0	2^{23}	2^{22}	2^{21}	....	2^{11}	2^{10}	2^{9}	2^{8}	0	...	0	0	0		

normalization of Low_Param = low_param.2^8.

High_Param	0	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}
TH_i or FLOORH	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

(This sizing implies a normalization factor of 2^9=512.)

HY.High_Param	0	0	2^{22}	2^{21}	2^{20}	....	2^{10}	2^{9}	2^{8}	2^{7}	...	2^{-5}	2^{-6}	2^{-7}		
↓ PM=01																
HY.High_Param	0	2^{22}	2^{21}	2^{20}	....	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	...	2^{-6}	2^{-7}	0		
$HY_i.2^{15}$	0	2^{22}	2^{21}	2^{20}	....	2^{10}	2^{9}	2^{8}	2^{7}	0	...	0	0	0		

normalization of High_Param = low_param.2^9.

6.2.1.7 Tone identification - Thresholds and noise floors determination

A simulation of a signal at -25 dBm was performed to determine the set of thresholds and noise floors associated to each frequency. The approximation was:

$$min\ (a/b) = min(a)/max(b)\ ;\ max\ (a/b) = max(a) / min(b).$$

The results were as follows: $input=sin(2\pi f..n)$

Low band parameters:

- frequency: 697 Hz

  LY_{max}=2548 $LY1_{min}$=1719 with f =697 Hz $TL_1 = \frac{1719}{2548} \cdot 2^6 \cdot 2^8 = 11053$

  LY_{min}=2310 $LY1_{max}$=360 f=770;852;941 $FLOORL_1 = \frac{360}{2310} \cdot 2^{14} = 2553$

- frequency: 770 Hz frequency: 852 Hz frequency: 941 Hz

  $TL_2 = 9849$ $TL_3 = 8900$ $TL_4 = 7489$

  $FLOORL_2 = 2497$ $FLOORL_3 = 2155$ $FLOORL_4 = 2090$

High band parameters:

- frequency: 1209 Hz frequency: 1336 Hz frequency: 1477 Hz frequency:1633 Hz

  $TH_1 = 12509$ $TH_2 = 11151$ $TH_3 = 11359$ $TH_4 = 11571$

  $FLOORH_1 = 2984$ $FLOORH_2 = 3574$ $FLOORH_3 = 2948$ $FLOORH_4 = 2415$

In the routines implemented, all $FLOORL_i$ or $FLOORH_i$ were set to 4000 = max($FLOORL_i$, $FLOORH_j$).

6.2.1.8 Twist - Twist definition

During its transmission, a tone undergoes numerous amplitude variations. This results in some frequency bands being weakened in relation to others. For example, high frequencies loose a larger amount of their power on line than low frequencies. Furthermore, some particular frequencies or band of frequencies can be amplified during filtering (due to filters ripple). This effect is called **twist**. Obviously, the detector has to take it into consideration.

The CEPT requires a DTMF detector to support up to *6 dB* of twist on the input power

$$\Rightarrow \left| P_L - P_H \right| < 6dB$$

$-6dB < P_L - P_H < +6dB$, the equivalence in linear is $\frac{1}{4} < \frac{P_L}{P_H} < 4$.

6.2.1.9 Twist - Quantization

The manipulation of words during twist decision is shown below:

Both power estimators are coded on 16 unsigned bits.

Y	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2

The twist factors are quantified on 13 signed bits. This permits a faster manipulation and a smaller code (short immediate addressing). As both factors are less than one, they are normalized in Q12 on 13 bits. The sign is always zeroed (positive factors).

TWISTF	S	S	S	S	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}

The product lies on 32 signed bits.

Y*TWIST	S	S	S	S	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	.....	2^{-8}	2^{-9}	2^{-10}

In it is left shifted at its transition to the accumulator (PM=01).

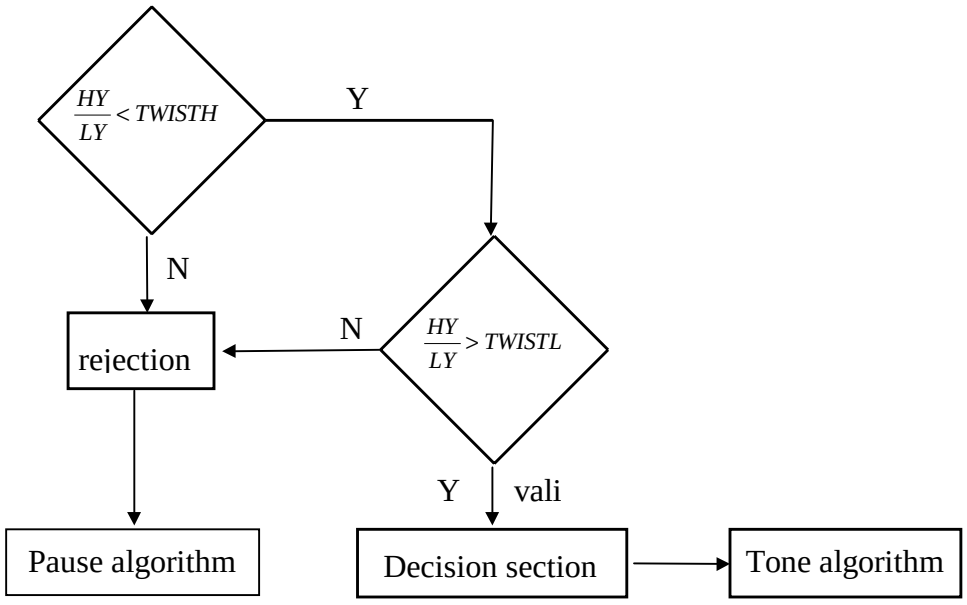
Y*TWIST	S	S	S	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	.....	2^{-8}	2^{-10}	0

To compare this product with the other estimator, it has to be left shifted by 13 bits.

$Y*2^{13}$	S	S	S	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	0	.....	0

6.2.1.10 Twist decision

The twist condition can be tested either with each power estimator or only with the band power estimators. The selected solution is to ensure that the low-band and the high-band estimators do not differ too much. It compares the ratio of low-band to high-band estimators with a certain tolerance scheme.



TWISTH and TWISTL depend on the digit already detected by previous tests. For each DTMF code there exist particular parameters (TWISTH,TWISTL). First TWISTH and TWISTL were the same for all DTMF code and had been determined, usinf Matlab: (2385,890). These values allowed us to start a test process with the aim of determining, digit by digit, the correct (TWISTHi,TWISTLi) pairs.

Here are the results:

Tone	TWISTH	TWISTL
1	3700	890
2	2200	1400
3	2300	1300
A	2900	990
4	3900	890
5	2300	1400
6	2385	1300
B	3000	1000
7	3900	890
8	2385	1400
9	2385	1200
C	3000	1000
*	3900	890
0	2385	1400
#	2385	1300
D	3000	1050

Having single values for TWISTH and TWISTL forces the detector to accept a larger twist range because the filtering introduces a non-linear attenuation of the signal: some frequencies are more attenuated than others. One can show that having a single pair for all digits puts the accepted twist range at about [-11dB ; +11dB].

6.2.2 Decision section

After tests, the detector must send its result to the host. The task of the previous section was to test signal power and its frequency components. This section analyses the signal timing and transmits its result via Acc and the content of a memory word – _OLD_OUT.

A word called **STATUS** is used to save the channel's own state .

bits 0 to 3 contain the previously detected digit.
bit 4 contains a flag: Pause Flag (P.F)
bit 5 contains a flag: Wait for a Pause Detection (W.P.D)
bit 6 contains a flag: Condition of Tone Detection (C.T.D)
bit 7 is unused.
bit 8-11 contain a pause counter (PAUSE_COUNTER)
bit 12-15 contain a tone counter (TONE_COUNTER)

bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

TONE COUNTER	PAUSE_COUNTER	free	C.T.D	W.P.D	P.F	DIGIT

If the Condition of Tone Detection (see 'Q.23 recommendation' chapter for its definition) is encountered, the C.T.D flag is set. When it is set, the host can read the DIGIT as the detected tone.

Having the Pause Flag set, also means that the previous 6ms of signal was not a DTMF signal.

Having WPD set, means that the decision process is waiting for a Pause.

At the end of the test section, the process can be routed to either the TONE or the PAUSE algorithm. If the 6ms of analyzed signal – **also called a slice** – is not recognized as a tone, the process goes to the PAUSE algorithm; otherwise it goes to the TONE algorithm.

6.2.2.1 TONE algorithm

The first task of this algorithm is to determine whether the detector is waiting for a tone or not.

- If it is waiting for a tone (WPD = 0)

It will try to find 4 successive and identical (same detected digit) conclusions before setting CTD (the reason for '4' is explained in 'Resonator transient response' from paragraph '1-5) Results').

If the tone detection is preceded by a pause conclusion, the detector will start a *guard-time* analysis of it.

If it had detected a different digit from the previous one, the previous one would be forgotten, and the detector would focus on the *guard-time* analysis of the current digit.

When the detector set CTD (=>Acc!=0), host can read the digit (in _OLD_OUT word from CONST section) as the detected one, and the detector will go to the second step: the wait for a 40 ms of a non-DTMF signal – a pause.

- If it is waiting for a pause (WPD = 1)
  It will only reset the pause counter.

6.2.2.2 PAUSE algorithm

The first task of this algorithm is to determine whether or not the detector is waiting for a pause.

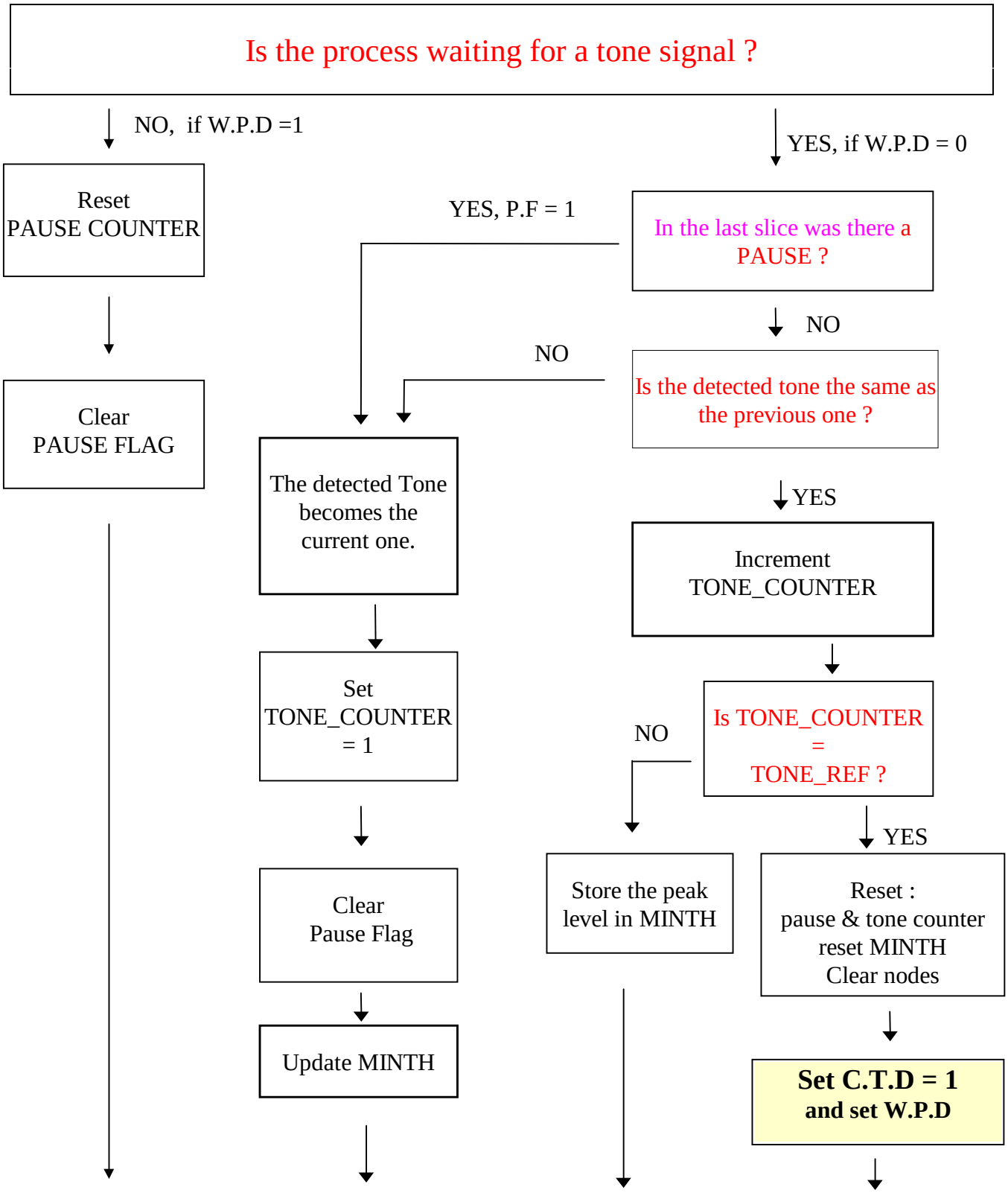
- If it is waiting for a tone
  It will reset TONE_COUNTER and set Pause Flag. So, in the next TONE algorithm process the current digit will become the one the detector would look for.
- If it is waiting for a pause
  It will increment PAUSE_COUNTER until it recognizes 40ms of pause (4 successive pause conclusions). Then it will clear WPD and the detector will reawait another tone detection (1st step of the detection).

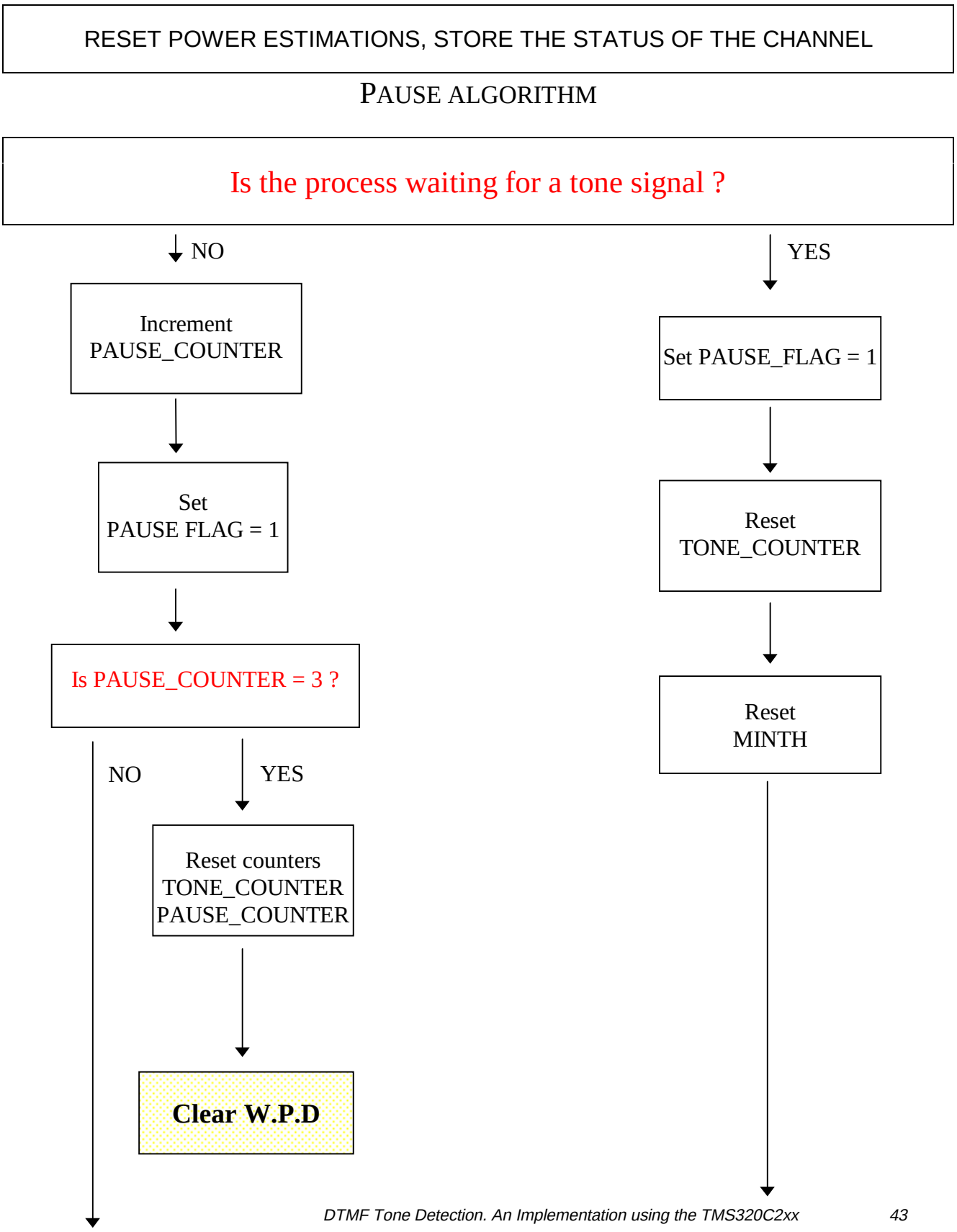
The detector, at the beginning of the process, waits for A PAUSE. (CTD=0, WPD=1, PF=1)

Flow charts of TONE and PAUSE algorithms are given in the next two pages.

NOTICE: CTD is always cleared at the beginning of BACKEND process.

TONE ALGORITHM




RESET POWER ESTIMATIONS, STORE THE STATUS OF THE CHANNEL
PAUSE ALGORITHM
Is the process waiting for a tone signal ?
NO
YES
Increment PAUSE_COUNTER
Set PAUSE FLAG = 1
Is PAUSE_COUNTER = 3 ?
NO
YES
Reset counters TONE_COUNTER PAUSE_COUNTER
Clear W.P.D
Set PAUSE_FLAG = 1
Reset TONE_COUNTER
Reset MINTH

RESET POWER ESTIMATIONS, STORE THE STATUS OF THE CHANNEL

6.3 Conclusion

All information about a channel state is stored in its associated STATUS word.

Users will notice that the detected digit becomes available only if C.T.D is set. In this case, the detected digit is in _OLD_OUT word from const section.

Two tests may be added:

1. **HY < MAXTHC & LY<MAXTLC ; to test if the signal is not too powerful**
2. **(LY+HY)< MAXTH ; to test if (LY+HY) < previous (HY+LY) +1dB**

7. Data Memory Organization

7.1 Requirements

The process needs two kinds of variables: the 'global' ones and the 'local' ones.

The 'global' variables are variables which do not depend on a specific channel: it can be threshold parameters or filter coefficients, for example. They are shared between all the channels.

The 'local' variables are the variables specially used for a particular channel: their values differ from one channel to another. They are useful to store a channel's own state, its filter nodes and its power estimations (LY .. HY4).

7.2 Global variables

They are allocated via a section called *'const'.*

Some of them are constants like filter coefficients, twist coefficients (...) while others are temporary variables like TEMP1-2, COUNT1, OLD_OUT, NEW_OUT (...).

Temporary variables are uninitialized but users do not have to initialize them:

- TEMP1-2, LYADD, HYADD, MINADD, NEW_OUT, OLD_OUT are always overwritten in the 'DETECT' process.
- while COUNT1 is cleared, just before being used.

Why have constants not been declared as '.set'?

In fact, you can change 'const' section by declaring SOME constants as '.set' and allow only temporary variables in this section. But if you decide to modify constant

values, during algorithm evaluation for example, you will have to assemble and link again: actually, you will do it every time you would like to perform a change. To allow constants such as .word xxxx permits you to change their figures by filling the right memory space in debugger tools, without having to assemble and link at every change.

If you decide to declare ALL constants as .set, you will have to adapt 'MPY' instructions, because they have 13-bits operands !

This section must be declared in your linker file and cannot be removed from 'detect.asm' file: the 'DETECT' routine uses both indirect and direct addressing modes when it calls variables from 'const' section.
THIS SECTION CANNOT CROSS PAGES. It must be loaded in a page but in the one you would like.

7.3 Local variables

These are used TO STORE A INDIVIDUAL CHANNEL VARIABLES like power estimations, pause and tone counters and others.

They don't need to be explicitly declared. It only depends on your channel management!

I decided to create the '_var' section and to declare it in mainC2xx.cmd and detect.asm because these made easier the initialization of '_DETECT' and '_INIT_VAR' input parameters.

The '_var' section is used in a single channel environment, it would be tedious and useless to create sections for each channel in a multi-channel environment.

The individual channel variables must be INITIALIZED before the beginning of the channel DTMF detection. All the space used must be cleared and STATUS, MINTH words must be loaded respectively with 0030h and (MINTHC+MINTLC). That is why we created a routine called 'INIT_VAR'. This routine allows users to do it simply, each time they handle a new channel (see program organization chapter for more information).

Space for a channel can cross pages: DP during '_DETECT' does not depend on the channel under consideration.

7.4 Conclusion

Memory requirement for 'const' and for each channel is specified in chapter 10.1.

There is no constraint about where individual channel parameters must be saved: it can cross pages and begin anywhere: it depends on your free memory space.

YOU ONLY HAVE TO:

- declare in your files 'const' section and be sure that it does not cross pages.
- initialize a channel variables before it is treated.

7.5 Tables

Below, you can find tables which show and describe in detail the global and particular variables.

DESCRIPTION OF CHANNEL DATA STRUCTURE

base+... = address	length (in words)	Data name	Access type	Description
0h	5	LY LY1 LY2 LY3 LY4	indirect via AR0	Contains power estimation of: LP1 output res697 output res770 output res852 output res941 output
5h	5	HY HY1 HY2 HY3 HY4	indirect via AR0	Contains power estimation of: HP2 output res1209 output res1336 output res1477 output res1633 output
Ah	1	SIGCNT	indirect AR0	Count the number of samples which have been filtered by the 'filtering section'.
Bh	1	STATUS	indirect AR0	Contains channel state. _OLD_OUT is loaded with it in the backend process (see at the end of this table for more information).
Ch	1	MINTH	indirect AR2	Contains the adaptative threshold.
Dh	2	resH4N	indirect AR1	delay line for: res1633
Fh	2	resH3N	indirect AR1	res1477
11h	2	resH2N	indirect AR1	res1336
13h	2	resH1N	indirect AR1	res1209
15h	2	HP2N	indirect AR1	HP2
17h	2	resL4N	indirect AR1	res941
19h	2	resL3N	indirect AR1	res852

1Bh	2	resL2N	indirect AR1	res770
1Dh	2	resL1N	indirect AR1	res697
1Fh	2	LP1N	indirect AR1	LP1
21h	2	HP1N	indirect AR1	HP1

STATUS format:

BIT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Tone counter				Pause counter				free	CTD	WPD	PF	previous Digit			

DESCRIPTION OF GLOBAL DATA ALLOCATED TO ‘const’ SECTION

address = base+...	Name	Access type	Description
0	HP1BO B1 A1 A2 B2	DMA	HP1 FILTER COEFFICIENTS
5	LP1B0 B1 A1 A2 B2	DMA	LP1 FILTER COEFFICIENTS
A	L1D L1C L2D L2C L3D L3C L4D L4C	DMA	LOW FREQUENCY RESONATOR COEFFICIENTS
12	HP2B0 B1	DMA	HP2 FILTER COEFFICIENTS

	A1 A2 B2		
17	H1D H1C H2D H2C H3D H3C H4D H4C	DMA	HIGH FREQUENCY RESONATOR COEFFICIENTS
1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2F	TL1 FLOORL1 TL2 FLOORL2 TL3 FLOORL3 TL4 FLOORL4 TH1 FLOORH1 TH2 FLOORH2 TH3 FLOORH3 TH4 FLOORH4	indirect via AR2	THRESHOLD PARAMETERS OF RESONATORS
30 31 32 33 34 35 35 36 37 38	TWISTH1 TWISTL1 TWISTH2 TWISTL2 TWISTH3 TWISTL3 TWISTHA TWISTLA TWISTH4 TWISTL4	indirect via AR2	TWIST PARAMETERS FOR EACH TONE

39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E	TWISTH5 TWISTL5 TWISTH6 TWISTL6 TWISTHB TWISTLB TWISTH7 TWISTL7 TWISTH8 TWISTL8 TWISTH9 TWISTL9 TWISTHC TWISTLC TWISTHE TWISTLE TWISTH0 TWISTL0 TWISTHF TWISTLF TWISTHD TWISTLD		
4F	NEW_OUT	DMA	If a tone has been detected, it would contain TEMPORARY the digit.
50	_OLD_OUT	DMA	STATUS of treated channel is always stored into _OLD_OUT at the beginning of 'backend' process.

51	TEMP1 TEMP2	DMA	Temporary variables
53	LYADD	DMA	Contains the complete 16bit address of LY from investigated channel. It is initialized every ‘backend’ process.
54	HYADD	DMA	Contains the complete 16bits address of HY from investigated channel. It is initialized every ‘backend’ process
55	MINADD	DMA	Contains the complete 16bits address of MINTH from investigated channel. It is initialized every ‘backend’ process.
56	COUNT1	DMA	This is a temporary counter.

8. Program Organization

Two routines are given into ‘detect.asm’ file:

- ‘_INIT_VAR’
- ‘_DETECT’

This chapter will explain what these programs are supposed to do, what their input parameters are and, if necessary, their outputs.

At the end of this chapter, we will briefly explain how the packages –mainC2xx (.asm&.cmd), and mainC5x (.asm& .cmd)– work.

8.1 ‘_INIT_VAR’

Before investigating a channel, the memory space reserved for it must be initialized.

- Its STATUS must be loaded with 0030h.
- Its MINTH must be loaded with (MINTHC+MINTLC)
- The rest of its space must be cleared (filter nodes, power estimators ...)

`'_INIT_VAR'` has been created to make it easier.

You have to specify the channel you want initialize by loading the start address of that space into AR2. Then you can call `_INIT_VAR`.

Example:

I have to manage 2 channels.

I know that each channel requires 35 words and I’d like to use:

- 0x0100 to 0x0122h for the first channel.
- 0x0870 to 0x0892h for the second one.

I would use '_INIT_VAR' like this:

```
.global _INIT_VAR
.
.
.
LAR      AR2,#0100h
CALL     _INIT_VAR
LAR      AR2,#0870h
CALL     _INIT_VAR
.
.
RETE
```

Notice that memory space reserved for channel 2 crosses pages 17 & 18. As 'Data memory' organization explains, this is not forbidden.

You have to call '_INIT_VAR', before '_DETECT', every time a new channel is examined.

8.2 '_DETECT'

This is the DTMF detector routine.

It has two input parameters:
- the sample to investigate
- the relevant channel.

YOU HAVE TO:
- Save the processor state (ST0, ST1) and save the contents of AR2,AR3,AR4 (they will be erased). AR5 will also be erased if you use test sections.
- Load AR2 with the lowest address of channel space location you want to treat.
- Store the new sample into TEMP1 from 'const' section.

Then you can call it.

For example: if 0x0100 to 0x0122 of data memory space is reserved for the channel we want to examine, we run:

```
; save AR2-4
; save ST0,ST1
.
.
LDP #0           ; load AR2
LAR AR2,#0100h

LACC DRR         ; load TEMP1
AND #FFFCh       ;forget bit 0-1 which are used for
                 ;communication with AIC
LDP       #TEMP1
```

```
SACL       TEMP1

CALL       _DETECT
```

The ‘_DETECT’ process MUST NOT be interrupted. Sometimes it is better to call it in the RINT interruption routine.

The channel memory space is pointed by AR2: AR4 & AR4 are used as secondary registers. So this space can cross pages without data page pointer troubles. Actually, during ‘_DETECT’, DP is always loaded with #CONST independently of the channel under examination.

How does one read the result of ‘_DETECT’ ?

The process transmits its result via the accumulator. Acc=0 => no detection : Acc!=0 => detection

When your process comes back from ‘_DETECT’, DP still equal to #CONST, so if acc!=0, you can directly read _OLD_OUT word — which had already been masked and so contains only the detected digit!

Remember that:

- bits 0-3 = tone digit.
  ```
  '1’ = 1h
  .
  .
  ‘9’ = 9h
  ‘0’ = ah
  .
  .
  ‘*’ = eh
  ‘#’ = fh
  ```
- bit 6 = Condition of Tone Detection bit.
  if ACC ≠ 0, the content of _OLD_OUT is available and you can read it as the detected tone.

Example:

```
.global _DETECT

CALL _DETECT
.
.
.
; each 24 CALL
BCND  DETECTION,NEQ
; nothing
; wait for another ‘_DETECT’ process
...
DETECTION
LACC  _OLD_OUT
```

```
    ;transmit to your channel manager the detected digit
```

8.3 About mainC2xx and mainC5x files ...

'detect.asm' can be linked with mainC2xx.asm via mainC2xx.cmd, or with mainC5x.asm via mainC5x.cmd.

By using mainC2xx(.asm .cmd), 'detect.asm' code can be implemented in the C2xx Simulator while mainC5x(.asm .cmd) allows users to use and test 'detect.asm' code with the C50 EVM.

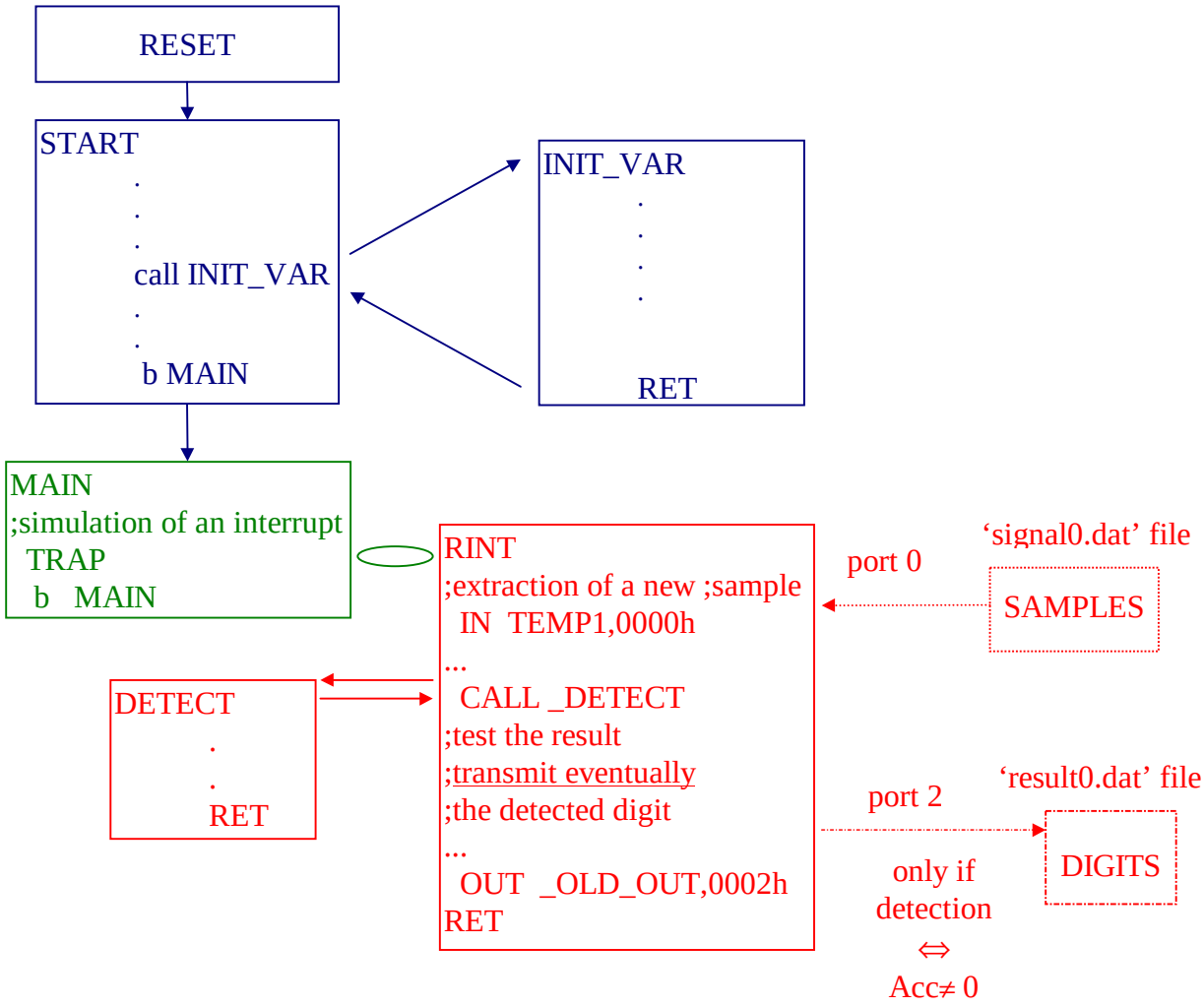
8.3.1 mainC2xx

in mainC2xx.asm

'vectors' section: this is a table which defines interrupt vectors.
'vec_def' section: this defines the 'START' and TRAP (also called *RINT*) routines.
'main' section: this initializes 'detect' input parameters. Samples must come from a file connected to 0x0000 I/O port.

in mainC2xx.cmd

'_var' section crosses pages. Programs are loaded in external memory while data memory is on-chip.



8.3.2 *mainC5x*

in mainC5x.asm

'vectors' section: this is a table which defines interrupt vectors.
'vec_def' section: this is an allocation for 'START', 'RINT' routines
'aicinit' section: this contains 'AIC_INIT' routine which had been built to initialize A/D converter from the EVM.
'main' section: this makes the DSP wait for an interrupt.

'START' will: initialize the DSP (mapping memory, processor mode, wait state generator)
initialize AIC by calling the 'aicinit' routine
initialize '_var' section by calling '_init_var'

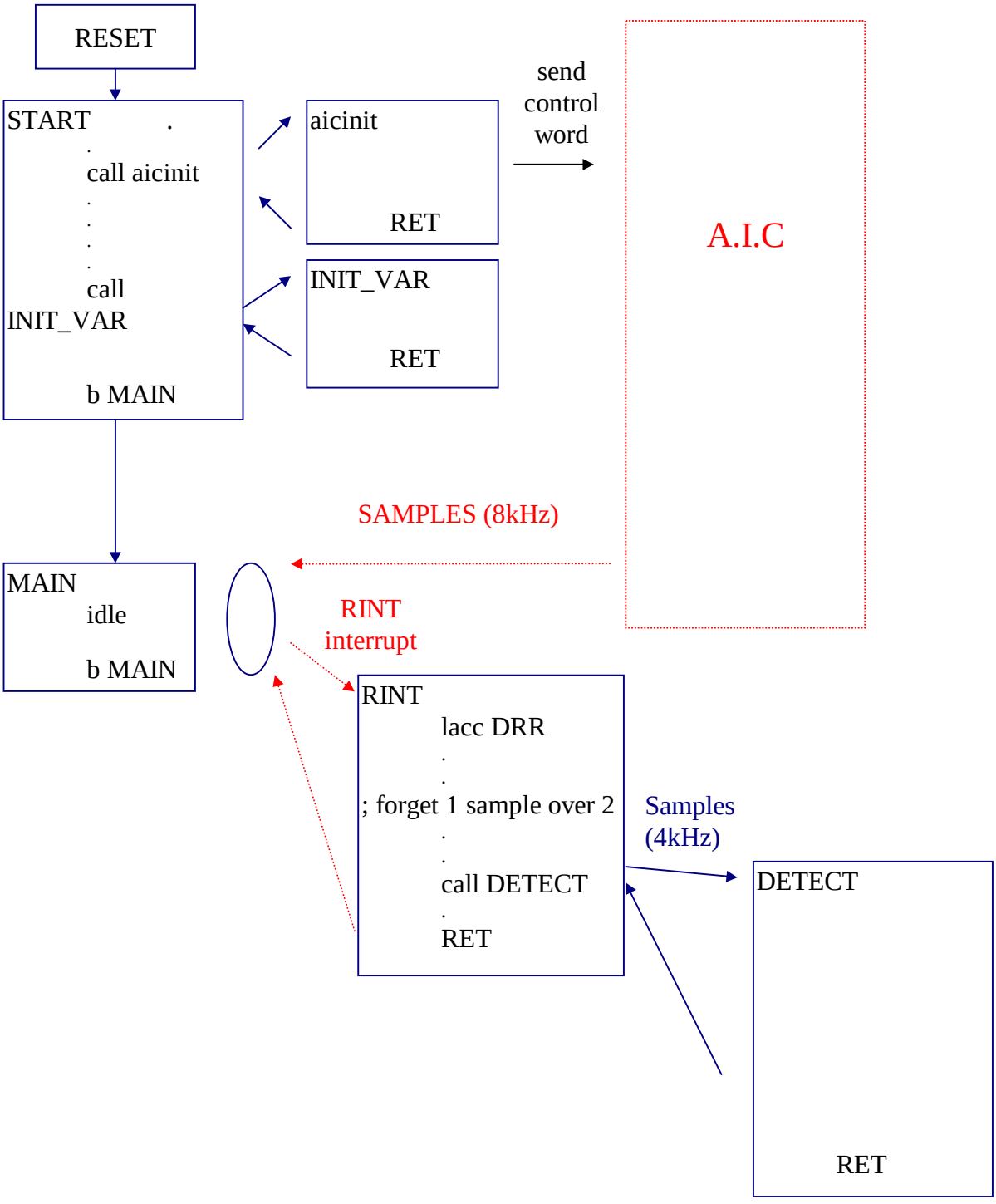
'RINT' will: load samples coming from the AIC into the accumulator
mask the 2 LSB of samples
transmit via DDR these samples (it's a sort of loop mode)

call ‘detect’ every one sample over two (8khz sampling rate becomes 4kHz)

‘aicinit’ will reset PA2 serial port (used for communication with AIC) and configure the A/D converter.

in mainC5x.cmd

same comments as the ones for mainC2xx.cmd


RESET
START .
call aicinit
call
INIT_VAR
b MAIN
aicinit
RET
INIT_VAR
RET
send
control
word
A.I.C
SAMPLES (8kHz)
MAIN
idle
b MAIN
RINT
interrupt
RINT
lacc DRR
; forget 1 sample over 2
call DETECT
RET
Samples
(4kHz)
DETECT
RET

9. Algorithm validation

9.1 Test Equipment

- a TMS320C5x EVM (Evaluation Module),
- a HP 33120A generator driven by HP Benchlink/Arb software,
- 'C2xx Simulator (SIM2xx.exe).

No telephone line has been interfaced with the EVM: so no DTMF signal coming from telephone network. No recorded tape could have been used. Therefore, the routine was checked with noised DTMF coming from HP generator.

9.2 Test procedure

9.2.1 Matlab simulation

Decision parameters like TWISTL1 (...) were first evaluated with Matlab.

Some functions have been created to test filters' real properties (with bit precision) and the filtering section. It allowed us to evaluate all test parameters introduced in various tests.

9.2.2 *DTMF signal generation*

HP software and generator were used to create tones.

Signal level (Umax):

It has been determined as if it were a sinusoidal function with a 600 ohms impedance.

Formula 1

$$P_{dB} = 10\log\left(\frac{U_{eff}^2}{1000 \times Z_0}\right) \text{with } Z_0 = 600\ \Omega$$

$$U_{eff}^2 = \frac{U^2{}_{\max}}{2}$$

Example: 0 dBm means 1.095V peak level.
to obtain a tone with low frequency component at 0 dBm, a sine wave was created with a peak voltage of 1.095V.

!! Warning !!

Using the HP software to create a tone involves the following approximation.

By adding two sine functions with X volts and Y volts peak levels respectively, one obtains a function with a peak at (X+Y) volts.

Xsin(2*pi*f1*t+P1)+ Ysin(2*pi*f2*t+P2)

= tone(f1,f2) with a peak at (X+Y) V.

This is absolutely WRONG but was necessary.

Thanks to Matlab, **this causes no trouble** at the required power: at most, the signal is 0.45 dBm more powerful per sine wave than it should be.

> We created the 16 DTMF signals at 0 dBm, and determined absolute peaks of each signal with Matlab. The approximation tells that the peak for a tone with 0 dBm for each sine component is 1.0954*2. And max(|1.0954*2-peaks from the 16 tones|)=|1.0954*2-1.0393*2|): this means we had set a sine level at 1.0954 while we would rather set it at 1.0393. So we added 0.45dBm for each sine wave.

Noise:

We added noise on each sine wave component of a tone: this noise is 20 dB less powerful than the sine wave level to which it has been added.

Example: tone ‘1’ at 0 dBm for each component is a tone with a peak at 2.409V and with a 17 dBm noise level.

> (0dBm)+(-20dBm) + (-20Bm)+(0dBm)
> = 1.095+ 0.1095 + 1.095 + 0.1095 V
> (thanks to the conversion formula 1)
> = 2.409 V

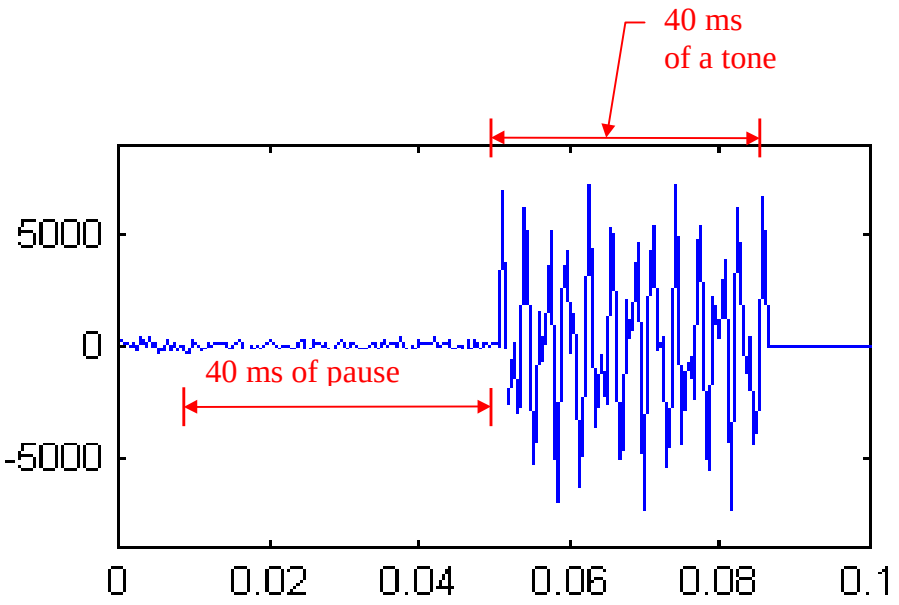
Noise is always added to DTMF signal.

Timing:

Signal is a succession of tone and pause: 40ms of tone, then 40ms of pause.

9.2.3 Criteria of tone recognition

- Expression ‘T signal’ is defined as 40ms of pause and 40ms of ‘T’ tone presence.



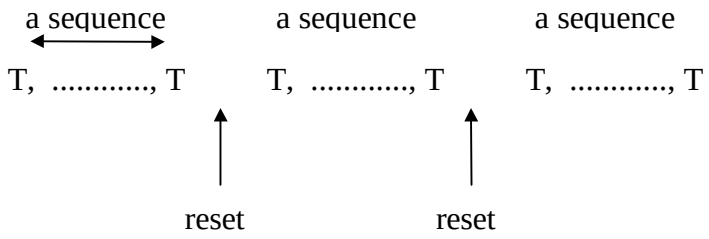
- A ‘T sequence’ is defined as a cascade of 40 ‘T signal’s.

T,T,.....,T,T

40 T signals
=
a sequence

The algorithm *‘is able to recognize the T tone’* if it is able to detect T EACH TIME the tone appears during 3 consecutive sequences.

By ‘consecutive sequences’, we mean that between sequences a DSP reset is performed.



9.2.4 Parameter modifications

TWISTH, TWISTL, TL, TH, MINTHC, MINTLC were adjusted during test to validate the algorithm, according to the Q.23 recommendation described in a previous chapter.

MINTHC, MINTLC:

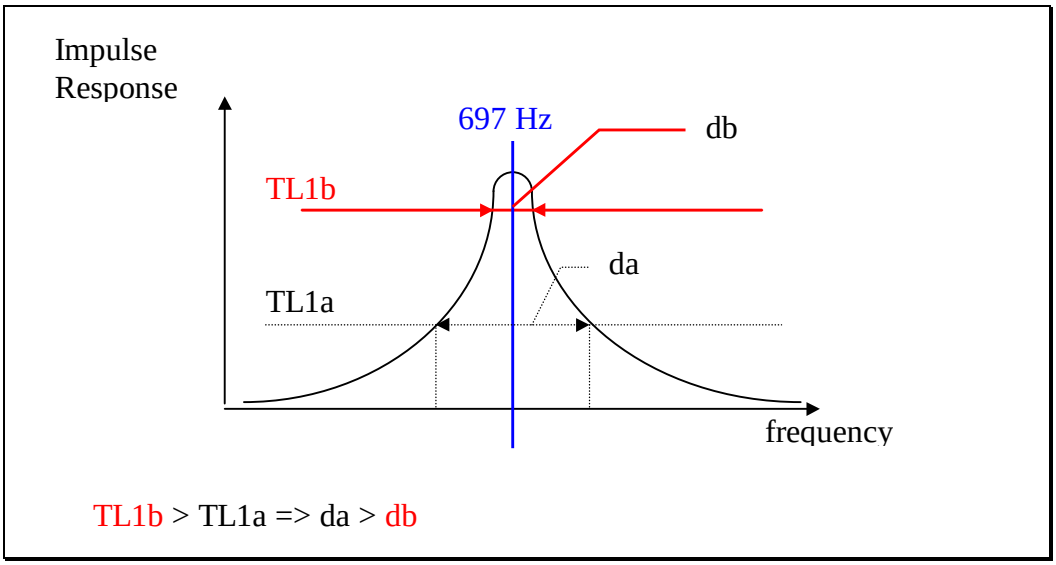
All tones (with low frequency power = high frequency power = -25dBm and with frequencies = standard ± 0 %) have been recognized during 4 successive sequences. Sometimes tones have been recognized at -24 dBm only, but a small decrease of MINT_C made them detectable at -25 dBm.

TL, TH:

To change these constant figures involve modification of frequency tolerance.

Example: to increase TL1 would reduce the band of frequencies around 697Hz that could be accepted.

Example: 697 Hz resonator

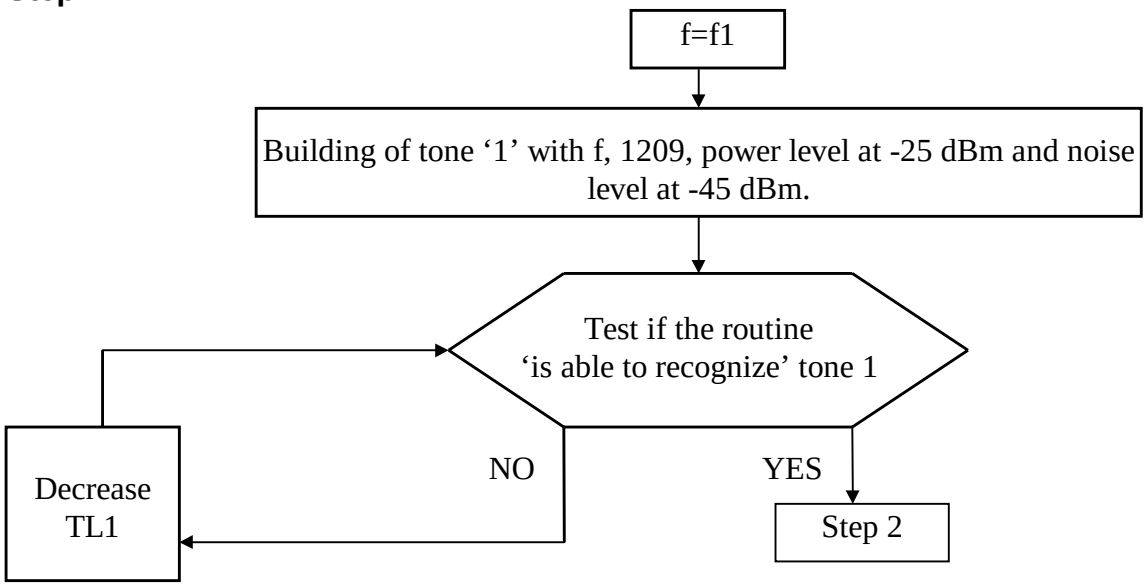


Flow chart of TL1 modifications (also for THx Tlx):

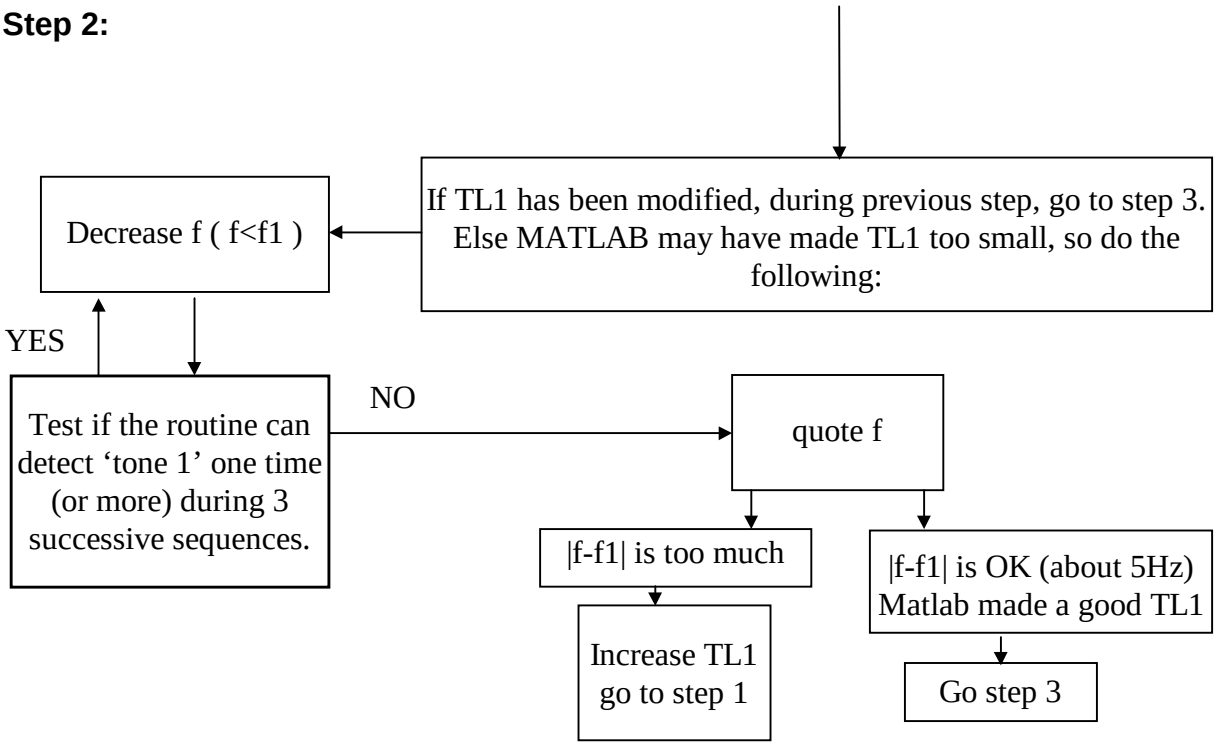
f1 = 697*(1-1.5/100) - 2 Hz
f2 = 697*(1+1.5/100) + 2 Hz

high frequency = 1209 Hz (I supposed that a 1209, 1336, 1477 or 1633 sine wave at -25 dBm would have about the same contribution on LY and LY1).

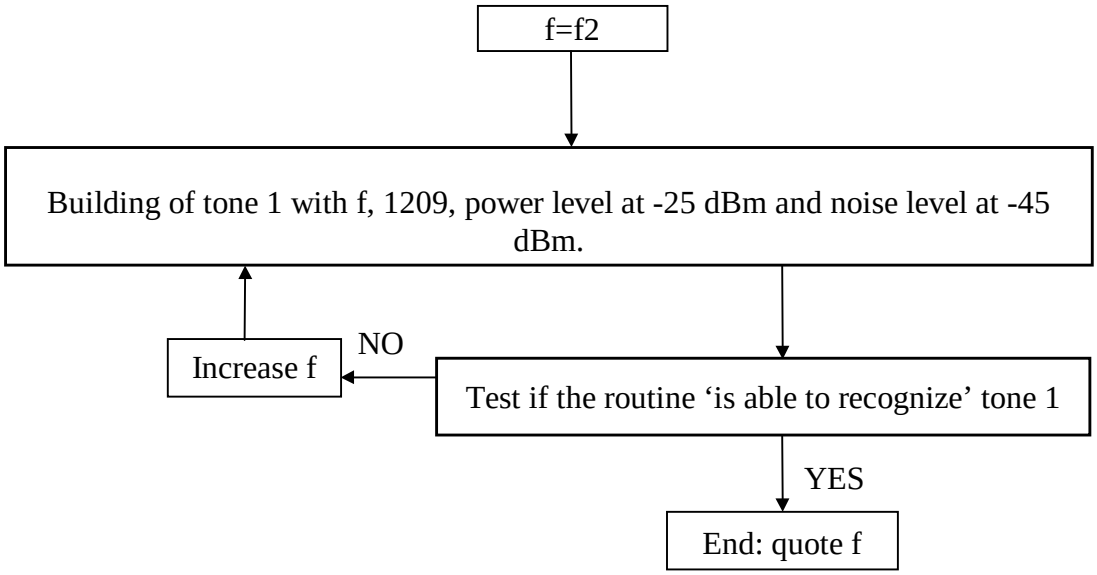
Step 1:



Step 2:



Step 3:



FLOORL, FLOORH:

Their initial figures(4000) are correct enough.

No modifications have been performed. But you can do it during other various tests, to make the routine more robust.

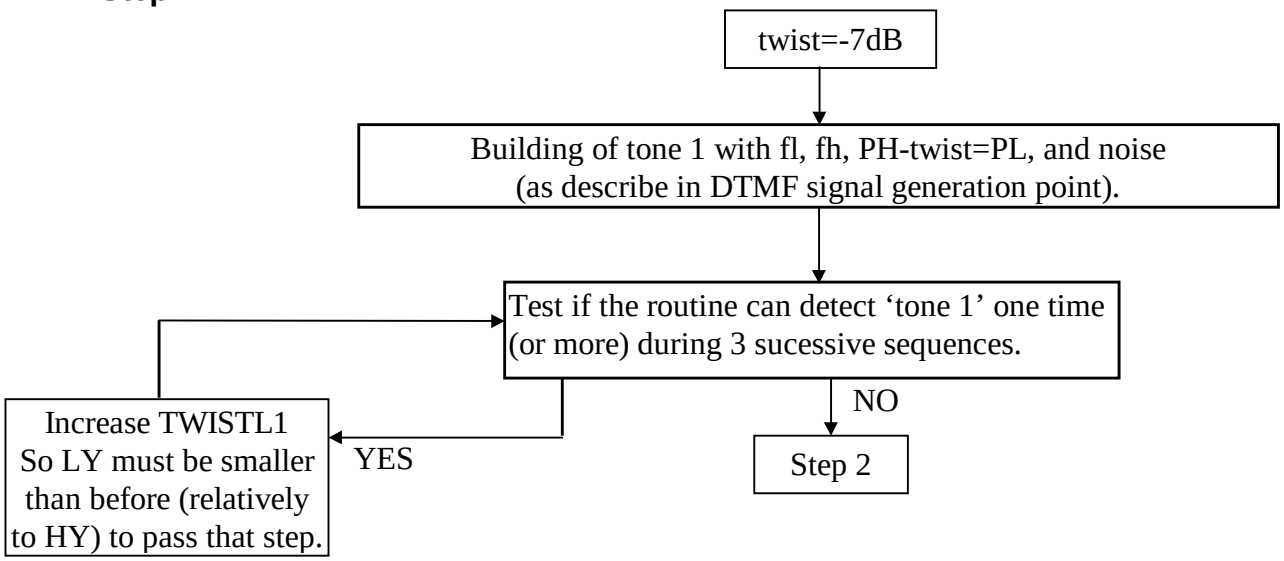
TWISTL,TWISTH:

The following example with TWISTL1 will show you how to adapt TWIST_x.

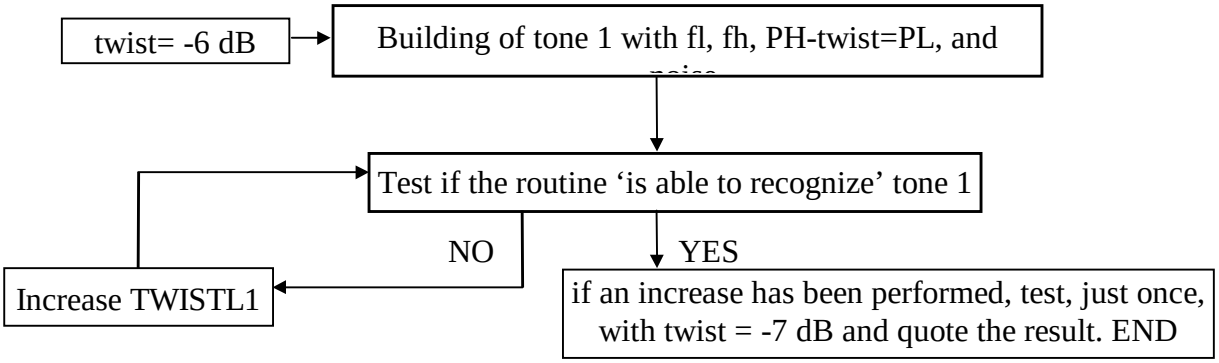
remember that conditions of acceptance were LY.TWISTL1 < HY.

Fl = 697 Hz.
Fh = 1209 Hz.
PL = power of low frequency component.
PH = power of high frequency component = -25 dBm.
Twist = -6 or -7 dB.

Step 1:



Step 2:



With these new figures, the routine has been submitted to a final test with the aim of determining 'the routine limits' in terms of gain, frequencies ... .

9.3 Results

Here are the results:

9.3.1 Minimum thresholds

Tone	Minimum power accepted for each component of the tone. (dBm)	Maximum power rejected. (dBm)
1, 2, 3, a 4, 5, 6, b 8, 0, #, d	-25	-26
7, 9, c, *	-25	-26.5 -27

9.3.2 Frequency acceptances

C [f1,f2] =]-00;f1[ ]f2;+00[
PL = power of low frequency component = -25 dBm.
PH = power of high frequency component = -25 dBm.

'No detection' means 'all of the tones -present in 3 successive sequences- were detected as pauses'.

Frequency	Specification	No detection	Able to recognize
697	± 1.79 %]684;710[	+ 2.44 / - 2.44 % C [680;714]	+2.15 / -2.00 % [683;712]
770	± 1.76 %]756;784[	+ 2.21 / - 2.21 % C [753;787]	+1.81 / -1.81 % [756;784]
852	± 1.73 %]837;867[	+ 1.99 / - 1.88 % C [836;869]	+1.64 / -1.76 % [837;866]
941	± 1.71 %]924;958[	+ 2.34 / - 2.34 % C [919;963]	+1.59 / -1.59 % [925;957]
1209	± 1.66 %]1188;1230[	+ 2.15 / - 2.07 % C [1183;1234]	+1.66 / -1.66 % [1189;1230]
1336	± 1.65 %	+ 1.87 / - 2.02 %	+1.65 / -1.65 % [1313;1358]

	]1313;1359[	C [1309;1361]	
1477	± 1.63%]1452;1502[	+ 1.96 / - 2.03 % C [1447;1506]	+1.62 / -1.62 % [1452;1501]
1633	± 1.62 %]1606;1660[	+ 2.02 / - 2.27 % C [1596;1666]	+1.78 / -1.78 % [1606;1662]

When 'No detection' were the result, we set PH=PL=-10dBm and retried. So we are sure that power were not the reason for this result.

9.3.3 Twists

PL=PH+twist

The lowest power of the two frequency components is always set to -25 dBm (except for *).

Example:

when you read PL=PH+6 , it implies that PH=-25dBm

and PL=-19 dBm and the noise components are (-45 dBm) + (-39 dBm)

Tones	accepted twist	rejected twist
1, 2, 3, 4 5, 6, 7 ,8 9, a, b, c d ,#	± 6	± 7
* (-24 dBm)	+6 / -5	± 7

Quote:

With twist = ± 7 dB, the highest power has been set to -10 dBm - as a consequence, the other level has been set to -17 dBm. So we were able to conclude that the unacceptable twist was the reason for the non-detection.

10. Memory space and MIPS required

10.1 Memory requirements

Section name	Type	Length (in words)
'init_var'	routine	17
'detect'	routine	548
'const'	variables & constants	87
for channels	variables	35 per channel
TOTAL		652+n.35

10.2 Cycle and Mips Requirements

10.2.1 'INIT_VAR'

Cycles: 58

No MIPS computation is meaningful: this routine consumes cycles just before the channel DTMF detection process.

10.2.2 DETECT'

Cycles

- 'filtering' section: 306 cycles (CALL and RET included)
- 'backend' section :344 cycles (BCND and RET included)

MIPS computation (notice: data memory location had been chosen to minimize MIPS)

The 'filtering' section is performed every 'DETECT' call while the 'backend' one is done every 24^{th}.

As a consequence, the averaged number of cycles needed per 'CALL' is:

$$\frac{(24\times 306+344)}{24} = 320 \text{ cycles}.$$

And, 'DETECT' is performed 4000 times per second (sampling frequency = 4000Hz) per channel, so the number of required MIPS is:

$$320 \times 4000 =$$

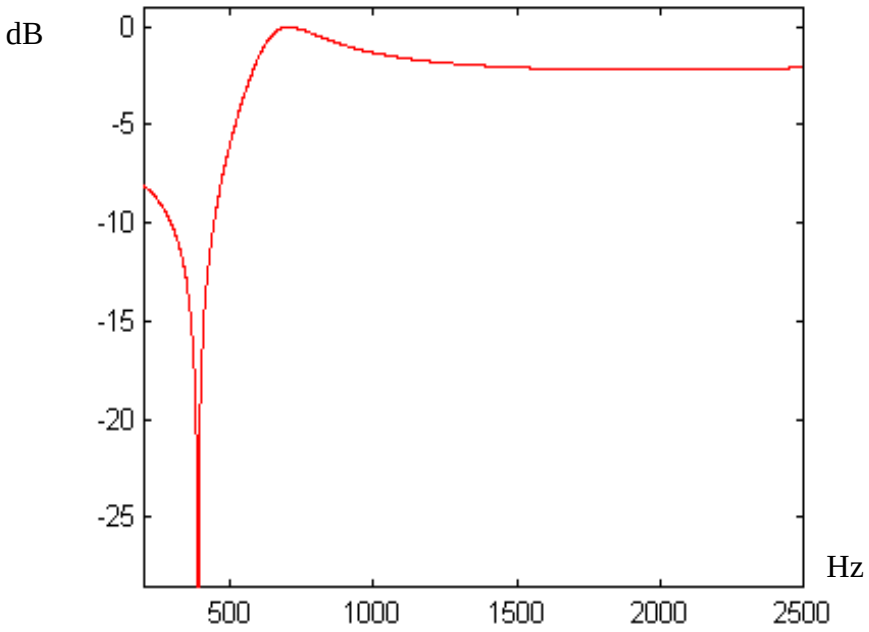
1.28 MIPS

Appendix A: Impulse Response Functions

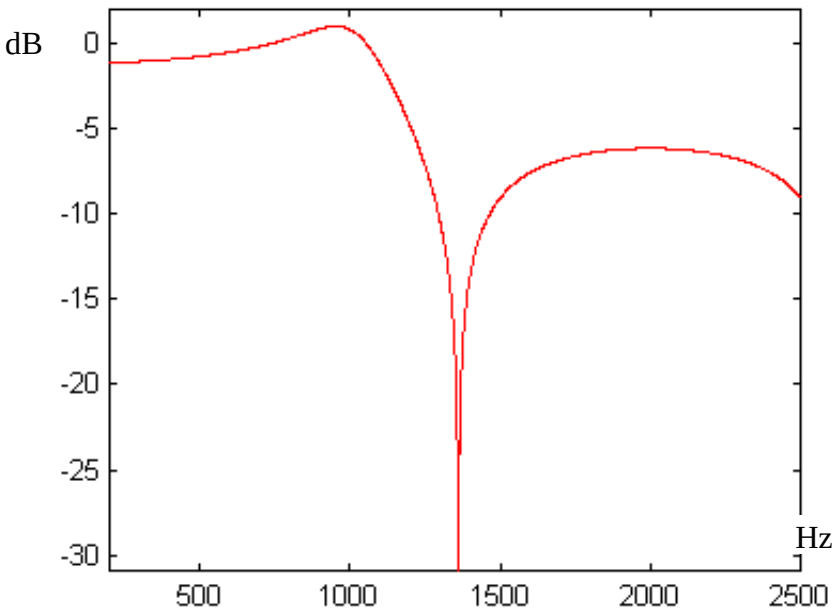
Number of samples: 4000

Sampling frequency: 4000 Hz

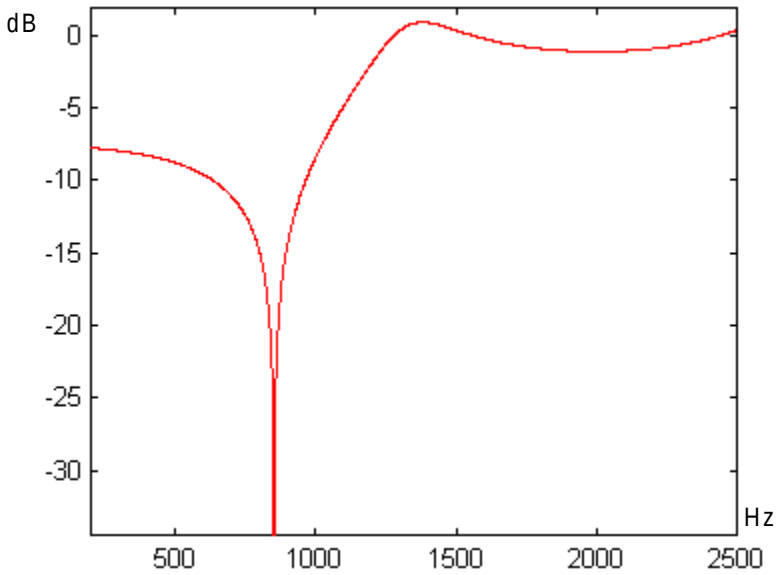
HP1 filter f_c (-3dB) = 560 Hz

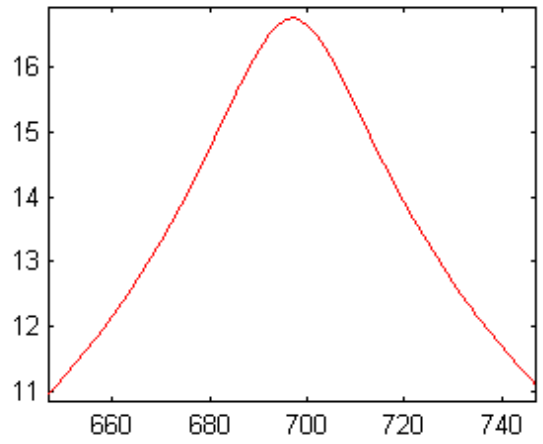
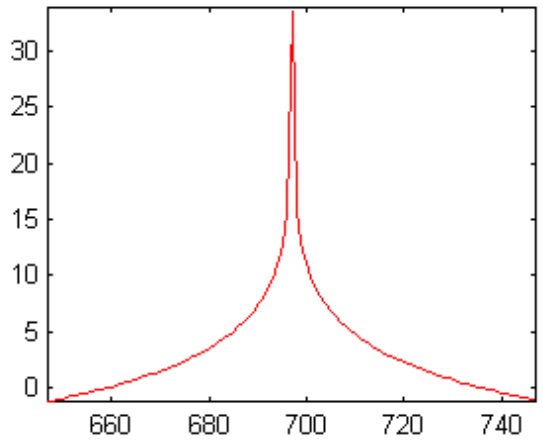


LP1 filter f_c (-3dB) = 1125 Hz



HP2 filter f_c (-3dB) = 1160 Hz



Frequency 697 Hz	
Implemented resonator	Goertzel equivalent
G_{max} = 16.75 dB 	G_{max} = 33.52 dB 

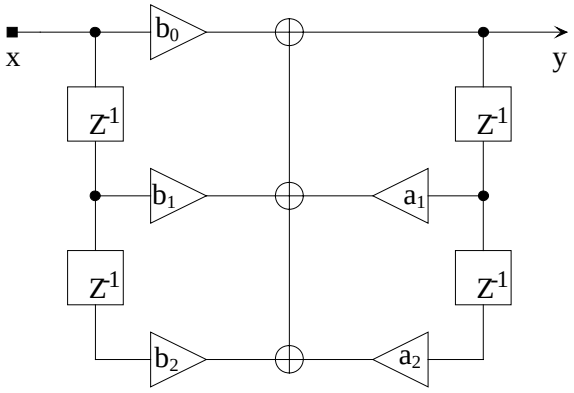
All other resonators (resp. Goertzel cells) have the same response as the resonator (resp. the Goertzel cell) tuned on 697 Hz. The only parameter, which is different from one filter to another, is Gmax:

	G_{max}	
Frequency (Hz)	Implemented resonator	Goertzel equivalent
770	16.24 dB	33.30 dB
852	15.76 dB	33.13 dB
941	15.3 dB	33 dB
1209	14.24 dB	33.24 dB
1336	13.98 dB	33.64 dB
1477	13.84 dB	35.64 dB
1633	14.07 dB	35.64 dB

Appendix B: Some refreshers on digital filtering

B.1 IIR biquad forms

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} + a_2 z^{-2}}$$



B.1.1 Direct form

$$y(n) = b_0\, x(n) + b_1 x(n-1) + b_2 x(n-2) + a_1 y(n-1) + a_2 y(n-2)$$

- 1 quantization point: *y(n)*
- 4 registers: *x(n-1) x(n-2) y(n-1) y(n-2)*

B.1.2 D-N structure

$$Y(z) = H(z).X(z) = \frac{N(z)}{D(z)}.X(z) = \frac{1}{D(z)}.N(z).X(z)$$

By introducing

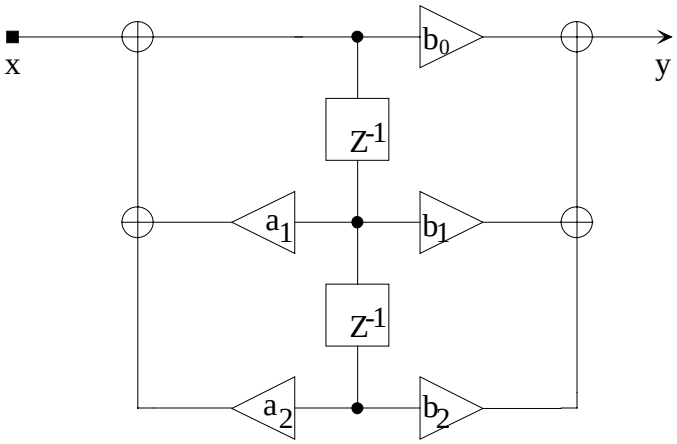
$$W(z) = \frac{1}{D(z)}.X(z)$$
$$Y(z) = W(z).N(z)$$

and so, two distinct difference equations appear:

$$\begin{cases} w_n = x_n + a_1.w_{n-1} + a_2.w_{n-2} \\ y_n = b_0.w_n + b_1.w_{n-1} + b_2.w_{n-2} \end{cases}$$

- 1 quantization point: w(n)
- 2 registers: w(n-1) w(n-2)

And the structure is:



B.1.3 N-D structure

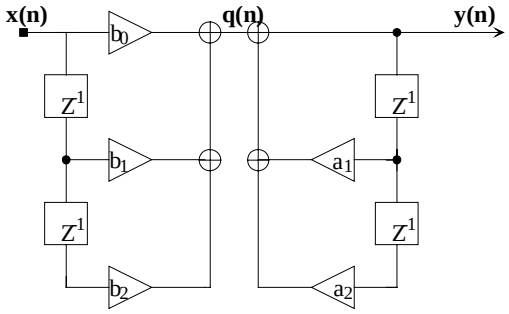
Reproducing the same reasoning as above, another structure results. This time, we introduce:

$$Q(z) = N(z).X(z)$$

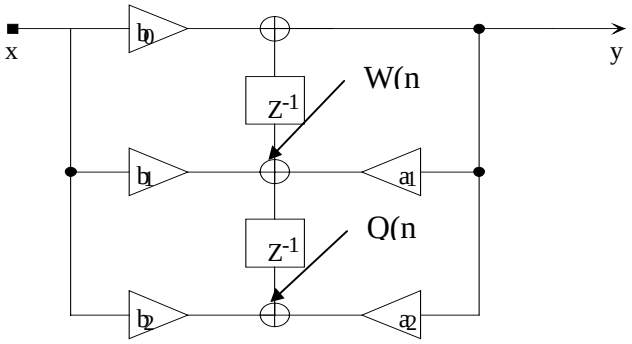
$$\Rightarrow Y(z) = \frac{1}{D(z)}.Q(z)$$

Thus, the difference equations satisfying this model are:

$$\begin{cases} q(n) = b_0.x(n) + b_1.x(n-1) + b_2.x(n-2) \\ y(n) = q(n) + a_1.y(n-1) + a_2.y(n-2) \end{cases}$$



This design can be easily modified to reduce by a factor of 2 the number of registers involved in the filter. So it becomes:



$$\begin{cases} y(n) = b_0 \cdot x(n) + w(n-1) \\ q(n) = b_2 \cdot x(n) + a_2 \cdot y(n) \\ w(n) = a_1 \cdot y(n) + b_1 \cdot x(n) + q(n-1) \end{cases}$$

2 quantization points: w(n) q(n)

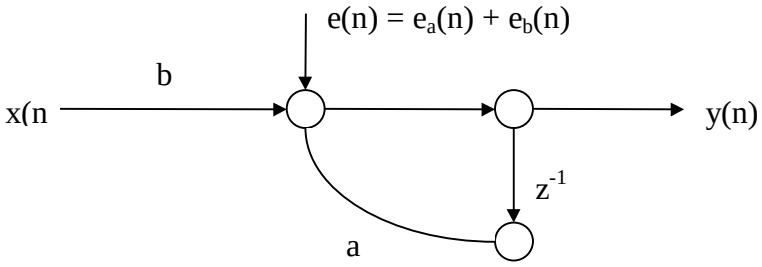
2 registers: w(n-1) q(n-1)

B.2 Effects of roundoff noise

Suppose we wish to implement a stable system having the system function:

$$H(z) = \frac{b}{1 - az^{-1}}$$

Here is the flow graph of the linear noise model for implementation in which products are quantized before addition:



The noise source:

- is a wide-sense stationary white-noise
- has a uniform distribution of amplitudes over one quantization interval
- is uncorrelated with the input to the corresponding quantizer, all other quantization noise sources, and the input to the system.

As shown in 'Discrete-time signal processing', by Oppenheim and Schafer, the noise variance at the output is:

$$\sigma_f^2 = 2.\frac{2^{-2B}}{12}.\frac{1}{1-a^2}$$

where B is the number of bits used for quantization.

The output noise variance increases as the pole at z=*a* approaches the unit circle. Thus, to maintain the noise variance below a specified level as |a| approaches unity, one must use longer word lengths.

B.3 Scaling methods of IIR Systems

The possibility of overflow is an important consideration in the implementation of IIR systems using fixed-point arithmetic. If one follows the convention that each fixed-point number represents a fraction (with possible assumed scale factor), each node in the network must be constrained to have a magnitude less than 1 to avoid overflow. If $w_k[n]$ denotes the value of the kth node variable and $h_k[n]$ denotes the impulse response from the input $x[n]$ to the node variable $w_k[n]$, then:

$$\left|w_k[n]\right| = \left|\sum_{m=-\infty}^{+\infty} x[n-m]\,h_k[m]\right| \qquad (3.1)$$

The boundary:

$$\left|w_k[n]\right| \le x_{\max} \sum_{m=-\infty}^{+\infty} \left|h_k[m]\right| \qquad (3.2)$$

is obtained by replacing $x[n\text{-}m]$ by its maximum value x_{max} and using the fact that the magnitude of a sum is less than or equal to the sum of magnitudes.

A sufficient condition that $|w_k| < 1$ is

$$x_{\max} < \frac{1}{\sum_{m=-\infty}^{+\infty} \left|h_k[m]\right|}$$

B.3.1 In any case

If 's_k' is the scaling factor for the k^{ith} node, it must satisfy:

$$s_k . x_{\max} < \frac{1}{\sum_{m=-\infty}^{+\infty} \left|h_k[m]\right|}$$

with $x_{max} = 1$ this becomes:

$$s_k = \frac{1}{\sum_{m=-\infty}^{+\infty} |h_k[m]|}$$

If 's' is the scaling factor for all nodes:

$$s = \frac{1}{\max_k \left[\sum_{m=-\infty}^{+\infty} |h_k[m]| \right]}$$

B.3.2 If a reset is performed after an M-sample treatment

$$s = \frac{1}{\max_k \left[\sum_{m=0}^{M-1} |h_k[m]| \right]}$$

B.3.3 If the input is a narrow band signal

The model of such a signal is: $x[n] = X_{max}\ cos(\omega_0\, n)$. In this case, the node variables will be:

$$w_k[n] = \left| H_k\left(e^{jw_0}\right) \right| . x_{\max} \cos\left(w_0 n + \angle H_h\left(e^{jw_0}\right) \right)$$

Therefore, overflow is avoided for all sinusoidal signals if:

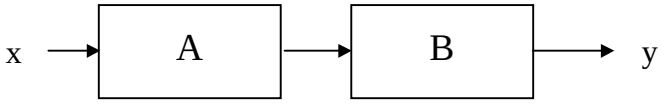
$$\max_{k,|w|\le\pi} \left| H_k\left(e^{jw}\right) \right| x_{max} < 1$$

or if the input is scaled by:

$$s = \frac{1}{x_{\max} . \max_{k,|w|<\pi} \left| H_k\left(e^{jw}\right) \right|}$$

B.3.4 With a cascade of filters

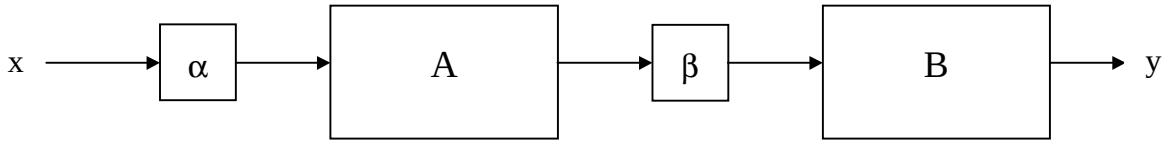
Example with two filters A and B:



step 1: One has to find the scaling factor of A, with one of the methods described previously

step2: if this factor is, say, α, one has to find the scaling factor of α-A-B. If this factor is called β then the system becomes:

(case of HP1-LP1-resonator)



Appendix C: About the ‘test’ section given with the code

Some code had been specially created to test the algorithm (on EVMC50 for example). It allows users to see conclusions built by the process.

This code is conditionally assembled

C.1 Test section validation

This can be done by setting the ‘TEST’ variable to ‘YES’:

```
TEST .set YES
```

‘TEST’ appears on the top of ‘detect.asm’ file.

Users have to validate this section or not.

```
(TEST     .set YES) or (TEST  .set NO)
```

C.2 How does it work ?

When the process detects a tone (40 ms of pause followed by 40 ms of tone signal), **the digit and the number of analyses between the previous detection and the current one are pasted in memory space.**

Three variables are added in the ‘const’ section:

COUNT2, COUNT3 and STOP_TEST.

Every time the process goes into the power analysis section (‘backend’):

- COUNT2 and COUNT3 are incremented.

Every time the process detects a tone

- it pastes the tone digit and COUNT3 in data memory space
- it clears COUNT3

When COUNT2 is equal to STOP_TEST, the routine is led to ‘stop_it’ label (located at the end of ‘detect’ section, after ‘reset_power’ label). A “b $” is performed at this location, so the DSP will wait until it is soft halted: it allows the memory space to be preserved.

When the DSP is halted, users can read results at the data memory space location (address of ‘varr’ section + 40 h).

C.3 Requirements

Memory: This depends on the number of detection !!

With STOP_TEST = 200 h, signal = cascade of (40ms of pause + 40ms of tone), 40 h is enough (1/2 page)

Register: This uses AR5 register. Results are pasted in indirect addressing mode, so the additional memory space does not have to be declared. AR5 content is saved in CONST

section and restored during the _DETECT process: so it can be erased by your own routines.

C.4 The use of COUNT2 and COUNT3

COUNT2

Actually, users can stop the DSP whenever they want. Without COUNT2 control, they must be careful about the memory management: the test process can erase an important memory area, and thus erase memory location used by any process.

With COUNT2, the DSP is halted on its own (if a “B$” command has been set at ‘stop_it’ label).

COUNT3

This is useful for the analysis of process conclusions.

Of course, the analysis depends on the input signal.

If this signal is a cascade consisting of a 40ms pause followed by 40ms of tone, and if it respects the Q.23 recommendation, COUNT3 must always be equal to dh or eh.

dh = 13*6ms = 78 ms

eh = 14*6ms = 84 ms

These are the times between two tones, whether the signal and the filtering process are well synchronized or not.

Appendix D: Calling the routines from C environment

The routine names have been underscored in order to be identified as being from a C environment.

In the same way, OLD_OUT had been underscored, too, becoming _OLD_OUT.

D.1 How to activate the C compatibility of these routines.

This is done by setting the assembly constant “C_compatibility” low or high. This constant is declared before variable and routine declarations, at the top of detect.asm file.

```
activate C environment => C_compatibility      .set    YES
disable  C environment => C_compatibility      .set    NO
```

Some sections are conditionally assembled: this condition is a function of “C_compatibility” setting.

D.2 Routine C prototypes.

D.2.1 _DETECT

prototype:

int DETECT (int,int)

input:

int from the left: it must contain the channel memory space base address.
int from the right: it must contain the sample to treat.

output:

if 0, it means no detection.
else it means detection and the detected digit is stored in _OLD_OUT.
The result of the function is physically stored in the accumulator.

declaration into C files:

```
extern int DETECT(int,int);     /* (WITHOUT UNDERSCORE) */
extern int OLD_OUT;
```

D.2.2 _INIT_VAR

prototype:

void INIT_VAR (int)

input:

it must contain the channel memory space base address.

output:

none.

declaration into C files:

extern void INIT_VAR(int); /* (WITHOUT UNDERSCORE) */

D.3 Conventions performed by the called routines.

_DETECT and _INIT_VAR respect conventions of C called functions;

- do not modify AR0,AR1,AR6,AR7
- return value in the accumulator.

For information: PMST is modified

AR2,AR3,AR4 are used as local pointers, so their contents are modified **but not restored**.

AR5 is used as a local pointer only if the test section is enable (TEST .set YES).

D.4 MIPS modification.

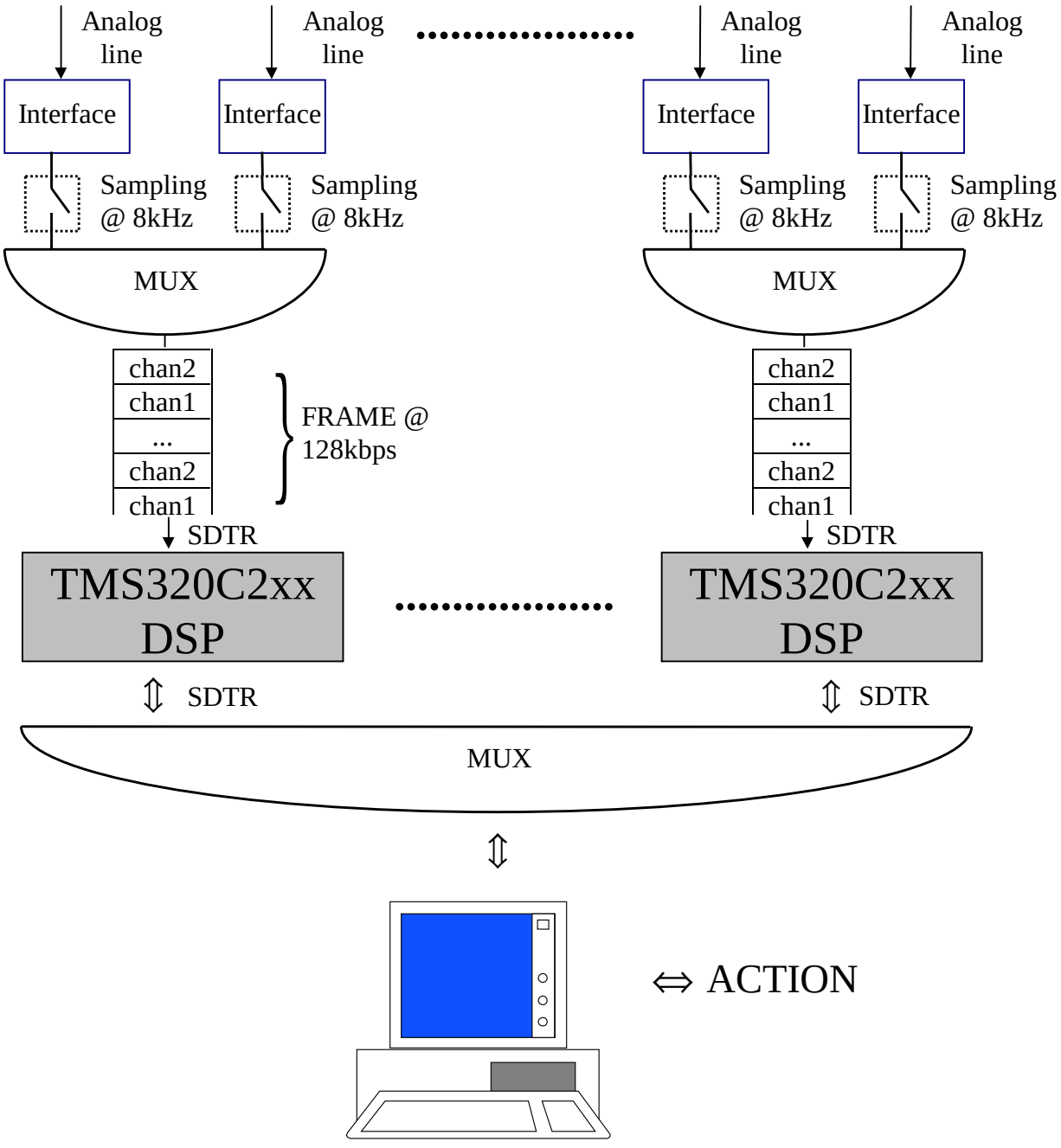
The filtering process needs 7 cycles more.
The analysis section needs only one cycle more (MAR *,AR1 near “reset_power” location)
So the process needs 4000*(24*7+1)/24 MIPS more.
As a consequence, the 1.27 needed MIPS becomes **1.28 MIPS.**

1.27 MIPS → 1.28 MIPS

Appendix E: Multichannel management example

E.1. Context

The DSP is used to perform detection on only 2 channels (for example): another system would provide channel management. The DSP and the 'manager' would communicate via the serial port.



E.2 DSP code example for simulator in C language.

Two memory space locations are first defined: 0x0200h for channel 1
0x0200h+35 for channel 2

Reset routine: 1- Initialization of C environment.

Main routine: 1- Initialization of channels.
2- Then the process calls, indefinitely, RINT (this acts as RINT interrupts).

RINT routine: 1- Extraction of a sample from signal0.dat or signal1.dat file.
2- Call of _DETECT.
3- Result management
-> if detection, then transmission of the detected digit via result0.dat or result1.dat file.
-> else nothing.

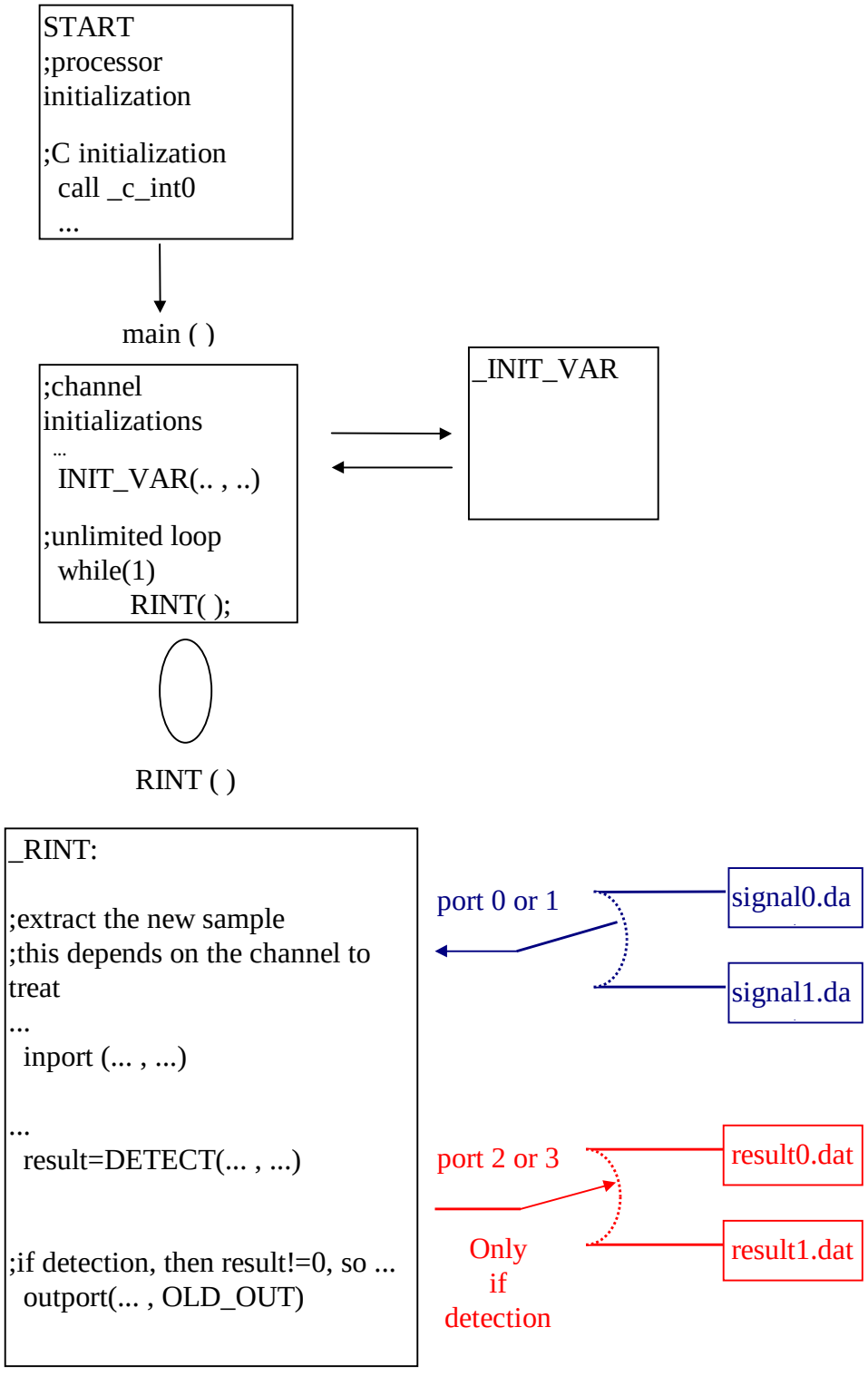
Here is cmain2xx.c:

```
#include "Ioports.h"

extern int DETECT(int, int);
extern int INIT_VAR(int);
extern int OLD_OUT;
void RINT(void);
#define chan_nb 2
int channel[chan_nb]={0x200,0x200+35};
int current_chan=0;

main(){
   int i;

   for(i=0;i<chan_nb;i++)
        INIT_VAR(channel[i]);

   /* simulation of an idle for a RINT */
   while(1)
        RINT();
}

void RINT(void){
   int result;
   int sample;

   inport(current_chan,&sample);
   result=DETECT(channel[current_chan],sample);

   if (result != 0){
        outport(current_chan+2,OLD_OUT);
   }
```

```
    current_chan++;
    if (current_chan == chan_nb)
         current_chan=0;
}
```

E.3 Block diagram.

2 channels are managed.



References

1. Agnès Delaurière, *DTMF Detection with TMS320C5x*, Texas Instruments, Vélizy, France, 1996
2. Alan V. Oppenheim, Ronald W. Schafer, *Discrete-Time Signal Processing,* Published by Prentice-All, Inc., A Division of Simon & Schuster, Englewood Cliffs, New Jersey, 1989
3. Andrew Bateman, Warren Yates, *Digital Signal Processing Design*, Computer Systems Series,Pitman Publishing
4. C.S Burrus, T.W Parks, *DFT/FFT and Convolution Algorithms,* A Wiley-Interscience Publication
5. Tim Massey and Ramesh Iyer, *DSP Solutions for Telephony and Data/Facsimile Modems*, Application Book, Texas Instrument Inc., 1997