Generation and Recognition of DTMF Signals with the Microcontroller MSP430

Robert Siwy
Texas Instruments Deutschland GmbH

SLAAE16
October 1997

TEXAS INSTRUMENTS

Contents part I

Contents part II

1 Introduction

The first part of the Application Report describes the generation of DTMF signals using the Microcontroller MSP430. Following an explanation of the most important specifications which are involved, the theoretical and mathematical processes will be discussed with which sinusoidal waveforms can be derived from square-wave signals, by making use of appropriate analog filters. Tested examples of software for generating square-wave signals for various timer configurations with the MSP430 are also provided. The chapter concludes with a circuit for deriving DTMF signals from the square-wave signals which have been generated.

2 The specification of DTMF signals

The abbreviation DTMF stands for "*D*ual *T*one *M*ulti *F*requency", and is a method of representing digits with tone frequencies, in order to transmit them over an analog communications network, for example a telephone line. During development, care was taken to make use of all frequencies in the voice band, in order to reduce the demands placed on the transmission channel. In telephone networks, DTMF signals are used to encode dial trains and other information. Although the method used until now to form dial trains from a sequence of current pulses is still the standard in Germany, the transmission time is too long and places an unnecessary loading on the network. In addition, many telecommunications services are only available with the use of tone dialing.

For DTMF encoding, the digits 0-9 and the characters A-D, */E and #/F are represented as a combination of two frequencies:

Frequency	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*/E	0	#/F	D

With this system, the column is represented by a frequency from the upper frequency group (Hi-Group: 1209-1633 Hz), and the line by a frequency from the lower frequency group (Lo-Group: 697-941 Hz). The tone frequencies have been chosen such that harmonics are avoided. No frequency is the multiple of another, and in no case does the sum or difference of two frequencies result in another DTMF frequency.

For the generation of a dial train in the Deutsche Telekom network, the specifications which follow must be met. These have been taken from the *Zulassungsvorschrift des Bundesamtes für Post und Telekommunikation, BAPT 223 ZV 5* [1] (Approval Specification of the Federal Office for Post and Telecommunications).

- The deviation of the actual frequencies generated from the nominal frequency must be a maximum of 1.8% during the dialling process

- The envelope of the dial train must conform to the waveform shown in Figure 1:

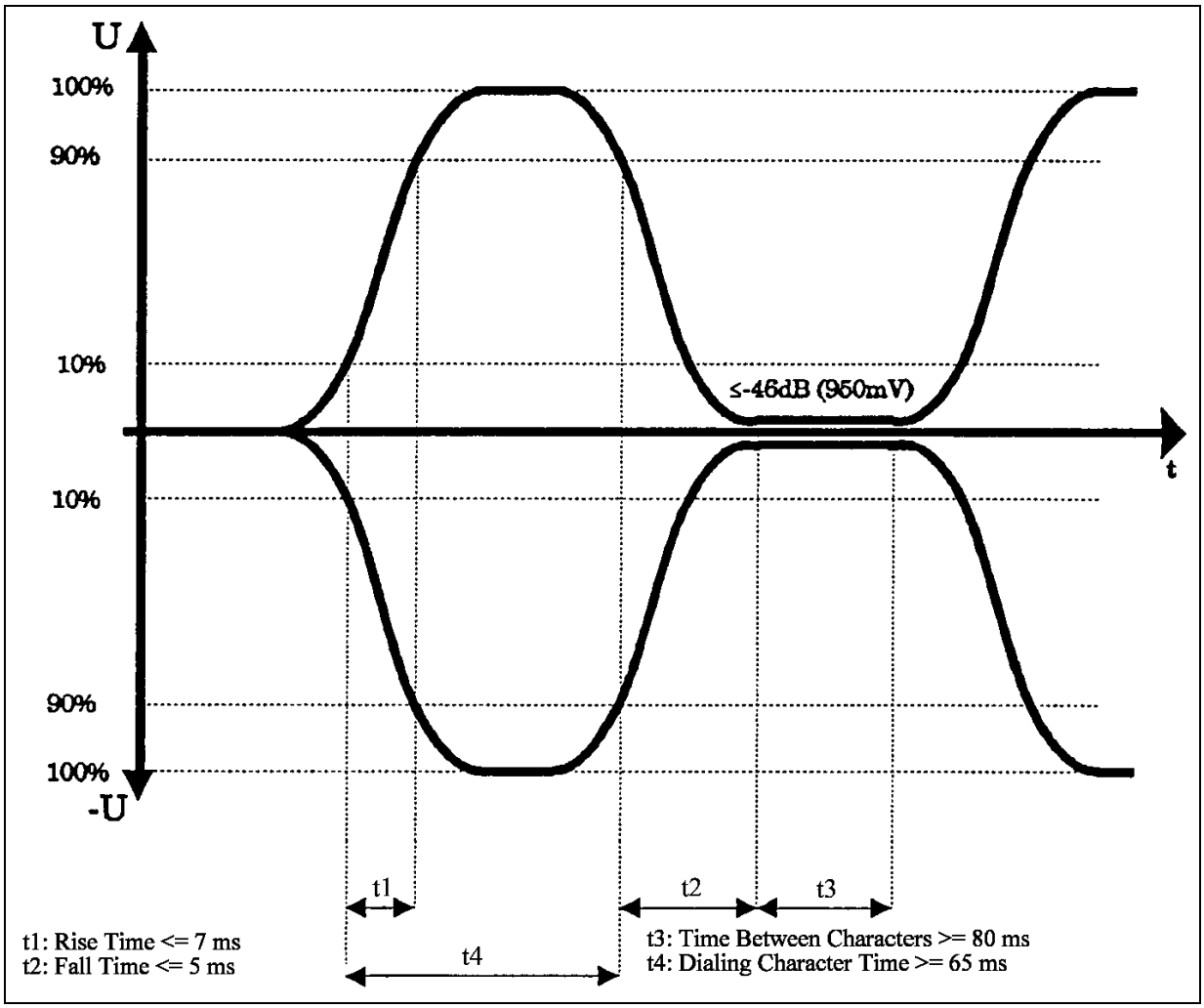

Figure 1: Timing of DTMF Characters

- The voltage levels must conform to the following values:

	Dialing Character Time	Time Between Characters	Minimum level in dB (950mV)		Maximum level in dB (950mV)	
			fu	fo	fu	fo
Automatic dialing, or manual dialing with automatic time limiting	65ms≤t≤100ms	80ms≤t≤6500ms	-16	-14	-10.5	-8.5
Manual dialing without time limiting	t≥65ms	t≥80ms	-16	-14	-13	-11

- The nominal voltage level of the higher of the two frequencies must be at least 0.5 dB higher (but no more than 3.5 dB higher) than the nominal voltage level of the lower of the two nominal frequencies, in order to compensate for line losses with long lines.

- In the frequency range of 250 Hz to 4600 Hz, the sum of the level of all frequencies which do not form a dial train must be at least 23 dB below the sum of the level of the existing dial train, and lie at least 20 dB below the level of the individual frequency of the dial train.

3 The generation of DTMF signals

As explained, DTMF signals are thus analog, and consist of two sine waves which are independent of each other. It is therefore not possible to generate them with only digital components. The digital signals must instead be converted by means of DACs (Digital-to-Analog Converters) and/or filters, into the desired sinusoidal waveforms.

3.1 Generation from square-wave signals

If DTMF signals are generated from square-waves, then the demands for hardware and software will be at a minimum.

Every recurrent waveform having a cycle duration of T can be represented by a Fourier series consisting of the infinite sum of individual sine and cosine waveforms [2], as follows:

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_0 \cdot \cos(n\omega_0 t) + b_0 \cdot \sin(n\omega_0 t)\right]$$

$a_0/2$ is the direct component of the signal. The partial component with the lowest angular frequency (ω_0) is termed the fundamental, and the others are known as overtones or harmonics.

A recurrent waveform which can be very easily generated with a microcontroller is the square wave, of which the Fourier series is as follows:

$$y(t) = \frac{\hat{y}}{2} + \frac{2\hat{y}}{\pi}\left[\sin(\omega_0 t) + \frac{1}{3}\sin(3\omega_0 t) + \frac{1}{5}\sin(5\omega_0 t) + \frac{1}{7}\sin(7\omega_0 t) + \cdot \quad \cdot \quad \cdot\right]$$

The shares which the individual frequencies have in the total signal can best be seen from the amplitude spectrum (see Figure 2):

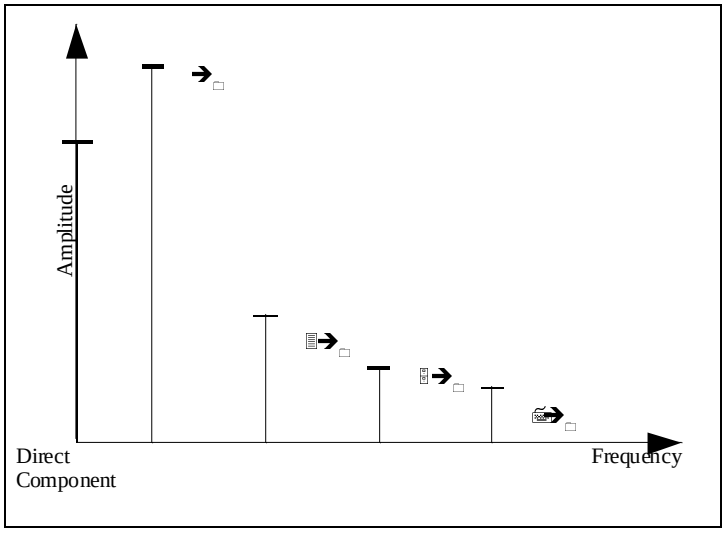

Figure 2: Amplitude Spectrum of a Square Wave

When an analog filter is used to attenuate the direct and harmonic components sufficiently strongly, a sinusoidal waveform with the same period as the square-wave will be obtained at the output.

3.2 Software for the generation of square-wave signals

The software for the generation of the square-wave signals must meet the following requirements:

- It must be able to generate two square-wave signals which are independent of each other.
- In order to separate the signals, two output pins are needed, which provide the outputs of the Hi-Group and the Lo-Group signals respectively.
- It must be possible to set the specific duration of the transmission of the signals over a wide range, of about 65 ms - 100 ms.

The MSP430 is provided with various timers which are suitable for generating square-wave signals. In the configuration ‘31x/‘32x, the 8-Bit and Timer Port timers are used, in order to generate both square-wave signals. This software is tested with a MCLK of 1.048 MHz. The Timer_A in the configuration ‘33x can generate both of the signals which are needed. The second software package uses this timer for the generation of the square-wave signals, and is also be tested with other MCLKs. The software for both configurations will now be described.

3.2.1 Generation of square-wave signals with 8-Bit and Timer Port timers

The flow diagram of the DTMF initialization routine is shown in Figure 3. In order to generate the two frequencies, the counters of the Timer Port and of the 8-Bit Timer are used. They are each provided with a programmable counting register, this being indispensable for setting the required frequencies precisely. If the Timer Port counter is cascaded to the 16-Bit timer, and allowed to operate at the system frequency MCLK, then the frequencies of the upper frequency group can be set very precisely. When there is an interrupt, the corresponding output pin is switched over, and both of the 8-Bit counter registers are reloaded. The values for this are read out of two RAM variables, in order to keep the internal registers available for other applications.

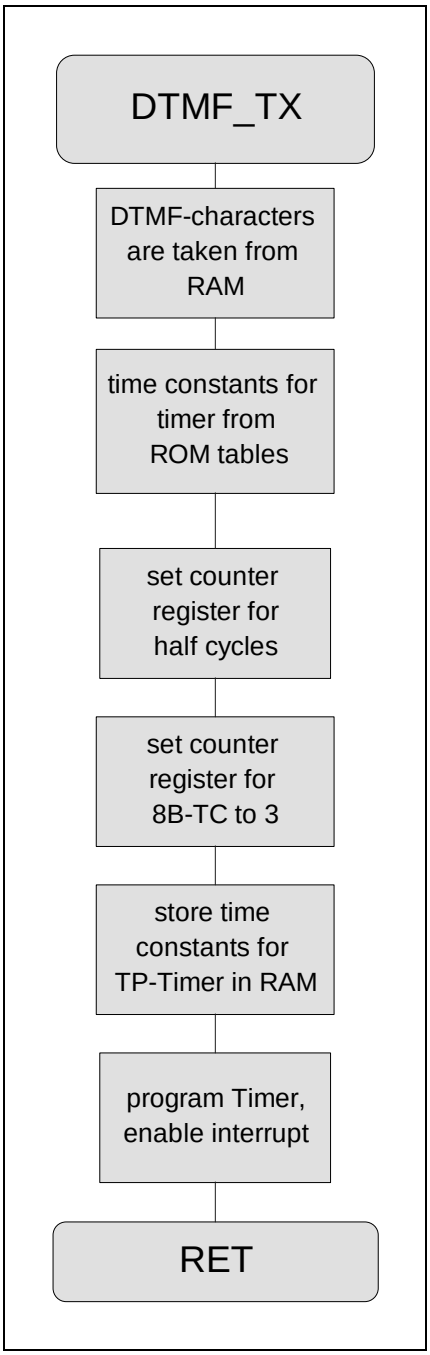

Figure 3: Flow Diagram of the DTMF Initialization Routine

The frequencies of the lower frequency group are generated by the 8-Bit timer. Since the counting register of this timer is only 8 Bit wide, only every third interrupt results in a change of the level at the desired output pin, in order for it to be still possible to generate the frequencies with the counter.

Two outputs of the timer port are used in order to output the two square-wave frequencies.

The initialization routine needs to be polled only once. After this has been done, the hexadecimal value of the number to be transmitted is read from a global RAM variable. After the pair of frequencies, consisting of low and high DTMF tones, has been generated from two tables, it is only necessary for both of the timers to be initialized and started. The duration of the transmission is monitored by counting the half cycles of the lower frequency, and read out

from an additional table. After this, the return to the polling function takes place. The corresponding interrupt routines perform the switching over of the port pins. This process is shown in Figures 4 and 5.

Whilst the timer port interrupt only reverses the logical level at the port pin and reloads the counter from the RAM, the 8-Bit timer interrupt is somewhat more demanding: the interrupts are counted by a counting register. The output can be changed only after three interrupts. In addition, every half cycle is included in the count. The output is interrupted when a specific number has been reached.

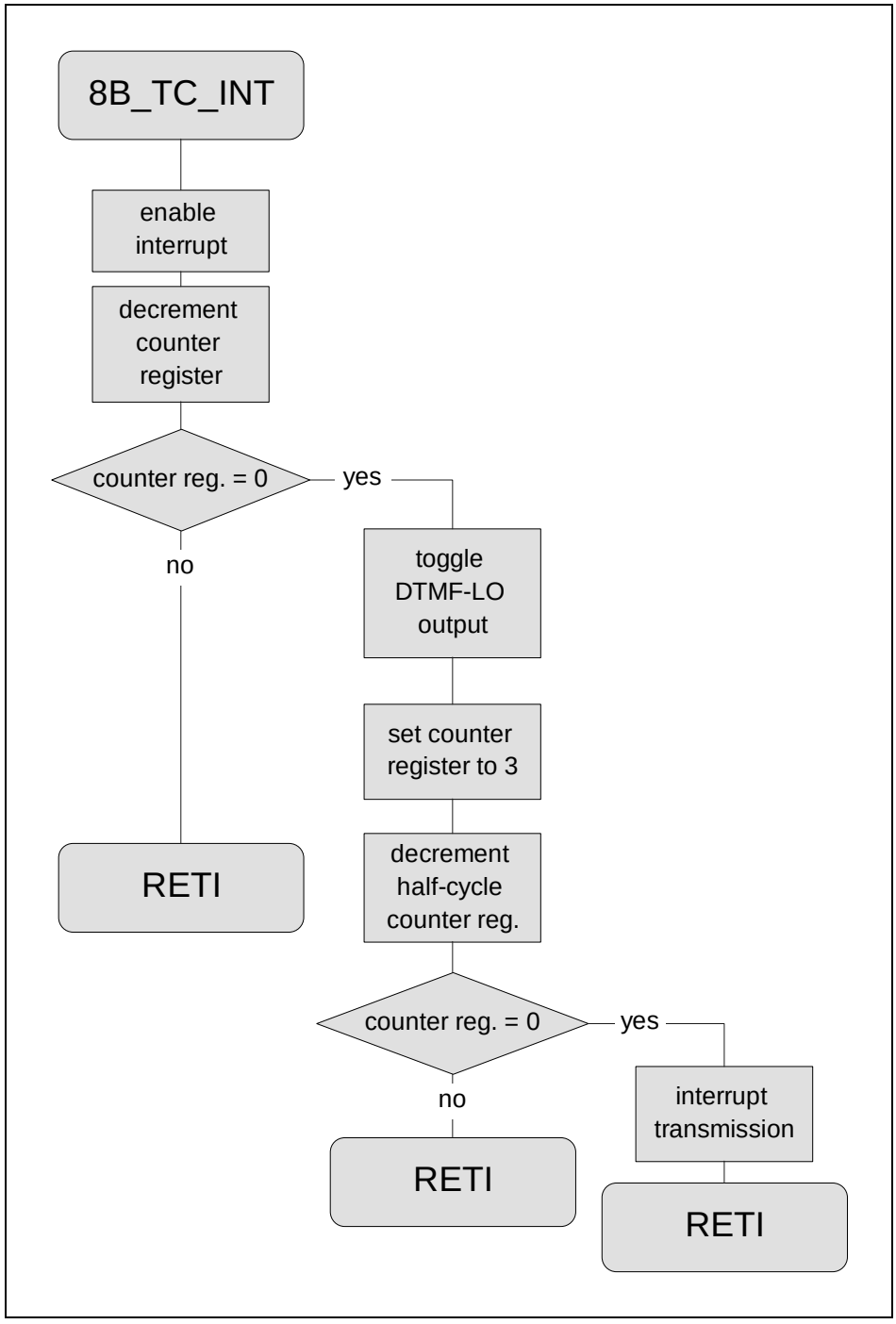

Figure 4: Flow Diagram des 8-Bit-Timer Interrupts (Lo-Group)

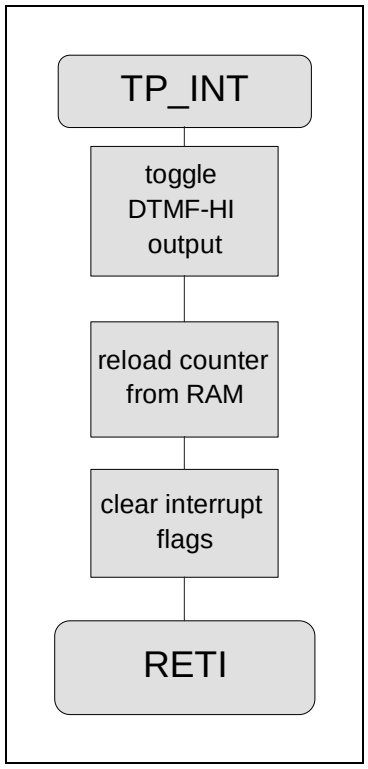

Figure 5: Flow Diagram of the Timer/Port Interrupts (Hi-Group)

```
; USER DEFINITIONS
FLLMPY    .equ  32             ;FLL multiplier for 1.048MHz
TCLK      .equ  FLLMPY*32768   ;TCLK: FLLMPY x fcrystal
DL        .equ 85         ;duration of DTMF-Signal (65..100ms)
LO_OUT    .equ 02h        ;Output Pin for low frequency
HI_OUT    .equ 04h        ;Output Pin for high frequency
RCOUNT    .equ r14        ;DTMF length counter
RTEMP     .equ r15        ;temp. register

               .global   DTMF_NR   ;global RAM-Variable
                                   ;for DTMF-Number (0..F)
; RAM DEFINITIONS

          .even
          .bss DTMF_TL   ;must be even adress!!!
          .bss DTMF_TH
          .bss DTMF_NR   ;global RAM-Variable
                         ;for DTMF-Number (0..F)
          .even

; HARDWARE DEFINITIONS FOR THE 8b-TIMER

TCCTL     .EQU 42H
TCPLD     .EQU 43H
TCDAT     .EQU 44H

;HARDWARE DEFINITIONS FOR THE UNIVERSAL-TIMER-PORT

TPCTL     .equ 04bh       ;Timerport Control
TPCNT1    .equ 04ch       ;TP Counter 1
```

```
TPCNT2    .equ 04dh      ;TP Counter 2
TPD       .equ 04eh      ;TP Data
TPE       .equ 04fh      ;TP Enable

          .text
; Tables with the DTMF frequencies: the table contains the
; number of MCLK cycles for a half period.

                    ;Table for high frequency
                    ;a correction value is added to
                    ;correct the latecy times
DTMF_HI   .word 0fffffh-(TCLK/(1336*2))+25    ;Hi-Freq for 0
          .word 0fffffh-(TCLK/(1207*2))+28    ;Hi-Freq for 1
          .word 0fffffh-(TCLK/(1336*2))+25    ;Hi-Freq for 2
          .word 0fffffh-(TCLK/(1477*2))+24    ;Hi-Freq for 3
          .word 0fffffh-(TCLK/(1207*2))+28    ;Hi-Freq for 4
          .word 0fffffh-(TCLK/(1336*2))+25    ;Hi-Freq for 5
          .word 0fffffh-(TCLK/(1477*2))+24    ;Hi-Freq for 6
          .word 0fffffh-(TCLK/(1207*2))+28    ;Hi-Freq for 7
          .word 0fffffh-(TCLK/(1336*2))+25    ;Hi-Freq for 8
          .word 0fffffh-(TCLK/(1477*2))+24    ;Hi-Freq for 9
          .word 0fffffh-(TCLK/(1633*2))+22    ;Hi-Freq for A
          .word 0fffffh-(TCLK/(1633*2))+22    ;Hi-Freq for B
          .word 0fffffh-(TCLK/(1633*2))+22    ;Hi-Freq for C
          .word 0fffffh-(TCLK/(1633*2))+22    ;Hi-Freq for D
          .word 0fffffh-(TCLK/(1207*2))+28    ;Hi-Freq for *
          .word 0fffffh-(TCLK/(1477*2))+24    ;Hi-Freq for #

                    ;Table for low frequency
DTMF_LO   .byte 0ffh-(TCLK/(941*2*3))   ;Lo-Freq for 0
          .byte 0ffh-(TCLK/(697*2*3))   ;Lo-Freq for 1
          .byte 0ffh-(TCLK/(697*2*3))   ;Lo-Freq for 2
          .byte 0ffh-(TCLK/(697*2*3))   ;Lo-Freq for 3
          .byte 0ffh-(TCLK/(770*2*3))   ;Lo-Freq for 4
          .byte 0ffh-(TCLK/(770*2*3))   ;Lo-Freq for 5
          .byte 0ffh-(TCLK/(770*2*3))   ;Lo-Freq for 6
          .byte 0ffh-(TCLK/(853*2*3))   ;Lo-Freq for 7
          .byte 0ffh-(TCLK/(853*2*3))   ;Lo-Freq for 8
          .byte 0ffh-(TCLK/(853*2*3))   ;Lo-Freq for 9
          .byte 0ffh-(TCLK/(697*2*3))   ;Lo-Freq for A
          .byte 0ffh-(TCLK/(770*2*3))   ;Lo-Freq for B
          .byte 0ffh-(TCLK/(853*2*3))   ;Lo-Freq for C
          .byte 0ffh-(TCLK/(941*2*3))   ;Lo-Freq for D
          .byte 0ffh-(TCLK/(941*2*3))   ;Lo-Freq for *
          .byte 0ffh-(TCLK/(941*2*3))   ;Lo-Freq for #

                    ;Table for signal length
DTMF_L    .byte     2*941*DL/1000  ;half periods for 0
          .byte     2*697*DL/1000  ;half periods for 1
          .byte     2*697*DL/1000  ;half periods for 2
          .byte     2*697*DL/1000  ;half periods for 3
          .byte     2*770*DL/1000  ;half periods for 4
          .byte     2*770*DL/1000  ;half periods for 5
          .byte     2*770*DL/1000  ;half periods for 6
          .byte     2*852*DL/1000  ;half periods for 7
```

```
          .byte     2*852*DL/1000  ;half periods for 8
          .byte     2*852*DL/1000  ;half periods for 9
          .byte     2*697*DL/1000  ;half periods for A
          .byte     2*770*DL/1000  ;half periods for B
          .byte     2*852*DL/1000  ;half periods for C
          .byte     2*941*DL/1000  ;half periods for D
          .byte     2*941*DL/1000  ;half periods for *
          .byte     2*941*DL/1000  ;half periods for #

;***************************************************************
;  DTMF-TX Subroutine for DTMF
;***************************************************************
DTMF_TX
     mov.b     DTMF_NR,RTEMP       ;save Number in temp. Reg.
     mov.b     DTMF_L(RTEMP),RCOUNT;save duration Counter
          ;prepare 8B-Timer for DTMF-Lo frequency
     mov.b     #0a8h,&TCCTL        ;configure Timer to MCLK
     mov.b     DTMF_LO(RTEMP),&TCPLD    ;prepare Pre-Load-Reg.
     mov.b     #000,&TCDAT         ;load Pre-Load in Counter
     bis.b     #008h,&IE1          ;enable TC-8b Int.
          ;prepare TP-Timer for DTMF-Hi frequency
     rla       r15                 ;* 2 for 16-Bit-Table
     mov       DTMF_HI(RTEMP),&DTMF_TL  ;save Hi-freq. as word
     mov       #003,RTEMP          ;counter for 8B-TC
     bis.b     #008h,IE2           ;enable TP-Int.
     mov.b     &DTMF_TH,&TPCNT2    ;Hi-Reloadvariable to TC2
     mov.b     &DTMF_TL,&TPCNT1;Lo-Reloadvariable to TC1
     bis.b     #080h,&TPD          ;enable 16Bit-Timer
     bis.b     #HI_OUT+LO_OUT,&TPE ;enable DTMF-Hi/Lo output
     mov.b     #090h,&TPCTL        ;enable Timer
     ret

;***************************************************************
;  Timer-Port Interrupt
;***************************************************************
TP_INT
     xor.b     #HI_OUT,&TPD        ;toggle Output for DTMF-Hi
     mov.b     &DTMF_TH,&TPCNT2    ;Hi-Reloadvariable to TC2
     mov.b     &DTMF_TL,&TPCNT1    ;Lo-Reloadvariable to TC1
     bic.b     #007h,&TPCTL        ;clear Flags
     reti

;***************************************************************
;  P0.1/8b-TC Interrupt
;***************************************************************
TIM_8B
     eint                          ;enable Interrupts
     dec       RTEMP
     jz        TOGGLE              ;3rd 8B-TC int. -> jump
     reti
TOGGLE
     xor.b     #LO_OUT,&TPD        ;toggle Output for DTMF-Lo
     mov       #003,RTEMP          ;counter for 8B-TC
     dec       RCOUNT              ;dec. durationcounter
     jz        DTMF_END            ;duration end -> jump
```

```
     reti
DTMF_END
     bic.b     #037h,&TPCTL        ;hold TP-Clock
     bic.b     #008h,TCCTL         ;hold 8B-Timer
     bic.b     #008h,&IE2          ;disable 8B-Timer-Int.
     bic.b     #003h,&TPE          ;disable output-pins
     reti

; INTERRUPT VECTOR ADDRESS

          .sect     "TP_VECT", 0ffe8h
          .word     TP_INT         ;Timer-Port

          .sect     "TIM_VECT", 0fff8h
          .word     TIM_8B         ;8b-Timer (P0.0 Int)
```

3.2.2 Generation of the square-wave signals with Timer_A

The following DTMF software routine only needs the Timer_A in order to be able to generate the two required square-wave frequencies. During assembling the corresponding timer values are calculated in order to be able to use the software independently of the MCLK which is employed. The length of the output signal is given with the value DL in milliseconds.

```
; Hardware definitions
;
FLLMPY    .equ  32             ; FLL multiplier for 1.048MHz
TCLK      .equ  FLLMPY*32768   ; TCLK: FLLMPY x fcrystal
DL        .equ  82             ; DTMF time ms (65..100ms)
STACK     .equ  600h           ; Stack initialization address
;
; RAM definitions
;
STDTMF    .equ  202h           ; Status Hi and Lo frequency
TIM32B    .equ  204            ; Timer Register Extension
LENGTH    .equ  206h           ; DTMF length counter
;
          .text 0F000h         ; Software start address
;
; Initialize the Timer_A: MCLK, Cont. Mode, INTRPT enabled
; Prepare Timer_A Output Units, MCLK = 1.048MHz (autom.)
;
INIT      MOV   #STACK,SP      ; Initialize Stack Pointer SP
          CALL  #INITSR        ; Init. FLL and RAM
          MOV   #ISMCLK+TAIE+CLR,&TACTL ; Define Timer
          MOV.B #TA2+TA1,&P3SEL ; TA2 and TA1 at P3.5/4
          CLR   TIM32B         ; Clear TAR extension
          BIS   #MCONT,&TACTL  ; Start Timer_A
          EINT                 ; Enable interrupt
MAINLOOP  ...                  ; Continue in mainloop
;
; A key was pressed: SDTMF contains the table offset of the
; two frequencies (0..6,0..6) in the high and low bytes
;
          MOV   &TAR,R5            ; For immediate start:
```

```
          ADD   FDTMFLO,R5          ; Short time offset
          MOV   R5,&CCR1            ; 1st change after 0.71ms
          MOV   R5,&CCR2            ; 1/(2x697) = 0.71ms
          MOV   #OMT+CCIE,&CCTL1    ; Toggle, INTRPT on
          MOV   #OMT+CCIE,&CCTL2    ; Toggle, INTRPT on
          MOV.B STDTMF,R5           ; Counter for 82ms
          RRA   R5                  ; # of low frequ. changes
          MOV.B DTMFL(R5),LENGTH    ; for the signal length.
          ...                       ; Continue background
;
; CCR0 interrupt handler (not implemented here)
;
TIMMOD0   ...
          RETI
;
; Interrupt handler for Capture/Compare Registers 1 to 4
;
TIM_HND   ADD   &TAIV,PC        ; Serve highest priority request
          RETI                  ; No interrupt pending: RETI
          JMP   HCCR1           ; CCR1 request (low DTMF frequ.)
          JMP   HCCR2           ; CCR2 request (high DTMF fr.)
          JMP   HCCR3           ; CCR3 request
          JMP   HCCR4           ; CCR4 request
;
TIMOVH    INC   TIM32B          ; Extension of Timer_A 32 bit
          RETI
;
; Low DTMF frequencies: TA1 is toggled by Output Unit 1
; Output changes of TA1 are counted to control signal length
;
HCCR1     PUSH  R5              ; Save used register
          MOV.B STDTMF,R5       ; Status low DTMF frequency
          ADD   FDTMFLO(R5),&CCR1 ; Add length of half period
          DEC.B LENGTH          ; Signal length DL elapsed?
          JNZ   TARET           ; No
;
; Yes, terminate DTMF signal: disable interrupts, Output only
;
          BIC   #OMRS+OUT+CCIE,&CCTL1   ; Reset TA1
          BIC   #OMRS+OUT+CCIE,&CCTL2   ; Reset TA2
TARET     POP   R5              ; Restore R5
          RETI                  ; Return from interrupt
;
; High DTMF frequencies: TA2 is toggled by Output Unit 2
;
HCCR2     PUSH  R5              ; Save used register
          MOV.B STDTMF+1,R5     ; Status high DTMF frequency
          ADD   FDTMFHI(R5),&CCR2 ; Add length of half period
          POP   R5              ; Restore R5
          RETI                  ; Return from interrupt
;
HCCR3     ...                   ; Task controlled by CCR3
          RETI
HCCR4     ...                   ; Task controlled by CCR4
          RETI
```

```
;
; Table with the DTMF frequencies: the table contains the
; number of MCLK cycles for a half period. The values are
; adapted to the actual MCLK frequency during the assembly
; Rounding assures the smallest possible frequency error
;
FDTMFLO   .word ((TCLK/697)+1)/2   ; Low DTMF frequency 697Hz
          .word ((TCLK/770)+1)/2   ;                    770Hz
          .word ((TCLK/852)+1)/2   ;                    852Hz
          .word ((TCLK/941)+1)/2   ;                    941Hz
FDTMFHI   .word ((TCLK/1209)+1)/2  ; High DTMF frequ.  1209Hz
          .word ((TCLK/1336)+1)/2  ;                   1336Hz
          .word ((TCLK/1477)+1)/2  ;                   1477Hz
          .word ((TCLK/1633)+1)/2  ;                   1633Hz
;
; Table contains the number of half periods for the signal
; length DL (ms). The low DTMF frequency is used for the timing
;
DTMFL     .byte 2*697*DL/1000  ; Number of half periods
          .byte 2*770*DL/1000  ; per DL ms
          .byte 2*852*DL/1000  ;
          .byte 2*941*DL/1000  ;
;
          .sect "TIMVEC",0FFF0h    ; Timer_A Interrupt Vectors
          .word TIM_HND            ; Timer Block 1..4 Vector
          .word TIMMOD0            ; Vector for Timer Block 0
          .sect "INITVEC",0FFFEh   ; Reset Vector
          .word INIT
```

A somewhat quicker solution is given below. It however requires somewhat more RAM capacity, because values derived from tables do not need to be calculated afresh each time, but instead are stored in the two RAM words DTMFLO and DTMFHI. These are read out from the Timer_A interrupt routines. The tables which are used are the same as in the previous example.

```
FLLMPY    .equ  32             ; FLL multiplier for 1.048MHz
TCLK      .equ  FLLMPY*32768   ; TCLK: FLLMPY x fcrystal
DL        .equ  82             ; DTMF time ms (65..100ms)
STDTMF    .equ  202h           ; Status Hi and Lo frequency
TIM32B    .equ  204            ; Timer Register Extension
LENGTH    .equ  206h           ; DTMF length counter
DTMFLO    .equ  208h           ; Half wave of low frequency
DTMFHI    .equ  20Ah           ; Half wave of high frequency
STACK     .equ  600h           ; Stack initialization address

          .text 0F000h         ; Software start address

; Initialize the Timer_A: MCLK, Cont. Mode, INTRPT enabled
; Prepare Timer_A Output Units, MCLK = 1.048MHz (autom.)
;
INIT      MOV   #STACK,SP      ; Initialize Stack Pointer SP
          CALL  #INITSR        ; Init. FLL and RAM
          MOV   #ISMCLK+TAIE+CLR,&TACTL ; Start Timer
          MOV.B #TA2+TA1,&P3SEL ; TA2 and TA1 at P3.5/4
```

```
          CLR   TIM32B          ; Clear TAR extension
          BIS   #MCONT,&TACTL   ; Start Timer_A
          EINT                  ; Enable interrupt
MAINLOOP  ...                   ; Continue in mainloop
;
; A key was pressed: STDTMF contains the table offset of the
; two frequencies (0..6,0..6) in the high and low bytes
;
          MOV   &TAR,R5         ; For immediate start:
          ADD   FDTMFLO,R5      ; Short time offset
          MOV   R5,&CCR1        ; 1st change after 0.71ms
          MOV   R5,&CCR2        ; 1/(2x697) = 0.71ms
;
; Fetch the two cycle counts for the DTMF frequencies
;
          MOV.B STDTMF+1,R5         ; High DTMF frequency
          MOV   FDTMFHI(R5),DTMFHI  ; Length of half period
          MOV.B STDTMF,R5           ; Low DTMF frequency
          MOV   FDTMFLO(R5),DTMFLO  ; Length of half period
;
                                    ; Counter for length
          RRA   R5                  ; Prepare byte index
          MOV.B DTMFL(R5),LENGTH    ; # of low frequ. changes
          MOV   #OMT+CCIE,&CCTL1    ; Toggle, INTRPT on
          MOV   #OMT+CCIE,&CCTL2    ; Toggle, INTRPT on
          ...                       ; to Mainloop
;
; CCR0 interrupt handler (not implemented here)
;
TIMMOD0   ...
          RETI
;
; Interrupt handler for Capture/Compare Registers 1 to 4
;
TIM_HND   ADD   &TAIV,PC        ; Serve highest priority request
          RETI                  ; No interrupt pending: RETI
          JMP   HCCR1           ; CCR1 request (low DTMF frequ.)
          JMP   HCCR2           ; CCR2 request (high DTMF fr.)
          JMP   HCCR3           ; CCR3 request
          JMP   HCCR4           ; CCR4 request
;
TIMOVH    INC   TIM32B          ; Extension of Timer_A 32 bit
          RETI
;
; Low DTMF frequencies: TA1 is toggled by Output Unit 1
;
HCCR1     ADD   DTMFLO,&CCR1    ; Add length of half period
          DEC.B LENGTH          ; DL ms elapsed?
          JNZ   TARET           ; No
;
; Terminate DTMF output: disable interrupts, Output only
;
          BIC   #OMRS+OUT+CCIE,&CCTL1   ; Reset TA1
          BIC   #OMRS+OUT+CCIE,&CCTL2   ; Reset TA2
TARET     RETI                  ; Return from interrupt
```

```
;
; High DTMF frequencies: TA2 is toggled by Output Unit 2
;
HCCR2     ADD   DTMFHI,&CCR2   ; Add length of half period
          RETI                 ; Return from interrupt
;
HCCR3     ...                  ; Task controlled by CCR3
          RETI
HCCR4     ...                  ; Task controlled by CCR4
          RETI
;
; Tables and interrupt vectors are identical to the previous
; example
```

3.3 Hardware for the generation of DTMF signals

As already mentioned, in the frequency range of 200 Hz to 4600 Hz the level of the transmission frequency must lie 20 dB above the level of all interfering signals. Since according to the specification the signals from the Highgroup and Lowgroup must have different levels, an individual filter is needed for each signal. The amplitudes and frequencies of all sinusoidal waveforms can be derived from the Fourier series.

When determining the cutoff frequencies in order to design the analog filters, two requirements must be met which result from [1]:

- Since it must be possible to combine every frequency from the Hi-Group with every frequency from the Lo-Group, the difference of level between the highest and the lowest frequency of a group may only be 3 dB.
- For the lowest frequency (f_1) of a group, the suppression of the harmonic ($3f_1$) must be at least 20 dB. The maintaining of this limit value is most critical for the lowest frequency of a group, since this frequency is furthest from the cutoff frequency of the filter.

The following equation [3] applies for the square of the absolute value of Butterworth low-pass filters of n^{th} order:

$$\left|A(f_1)\right|^2 = \frac{1}{1+\left(\frac{f}{f_g}\right)^{2n}}$$

This equation describes the behavior of the amplification of Butterworth low-pass filters as a function of frequency. The parameters f_g and n determine the cutoff frequency and order of the filter.

The order of the filter should first be calculated, this being needed in order to meet the requirements above.

In order to meet the first requirement, the ratio of the squares of the absolute values of the lowest and highest frequency of a group may only be 3 dB or ⑥2 :

$$\frac{|A(f_1)|^2}{|A(f_4)|^2} < \sqrt{2}$$

The second requirement will already have been met if the ratio of the squares of the absolute values of the frequencies f_1 and $3f_1$ is 10/3, since the harmonic in a square-wave signal is already 1/3 lower (see also Fourier series and Figure 2):

$$\frac{|A(f_1)|^2}{|A(3f_3)|^2} < \frac{10}{3}$$

Calculations with both Hi-Group and Lo-Group frequencies result in a filter with an order of n=1.15. A 2nd order filter, which can be constructed with an operational amplifier, would therefore suffice in order to meet the required limit values. If a 3rd order filter is used, then only two more components are required. In this way the sensitivity to tolerances can be reduced. Both requirements will be met if the cutoff frequencies of the filter lie within the following limits:

Lo-Group	fg>880Hz	fg<1418Hz
Hi-Group	fg>1527Hz	fg<2460Hz

If the cutoff frequency is at the lower limit, then the harmonics will be most effectively suppressed; however, the difference of level between the highest and lowest frequencies will then be 3 dB. With the highest possible cutoff frequency the difference of level is at a minimum, but harmonic suppression will then be only 20 dB.

When designing the filters, great care was taken to suppress harmonics, and the difference of level within a frequency group was fixed at 2 dB. As a result of this, the cutoff frequencies of the filters turned out to be 977 Hz and 1695 Hz. The suppression which resulted is thus considerably better than required. The difference of level within a frequency group is great enough to meet the required values, even if there is a shift of the cutoff frequency as a result of tolerances. When calculating component values, resistors were chosen to approximate to values which are available in the standard range E12.

After passing through the filter stages, two sine-wave frequencies are obtained which are separated sufficiently from those of the interference signals. In order to add both frequencies together, the circuit includes a subsequent adding stage.

With only three operational amplifiers and a few passive components, it is thus possible without much calculation effort to generate DTMF signals using a microcontroller.

For verifiing the aproximaten values, some runs with a simulation program are done. The effect of the filters can be predict very precise from the calculated frequency response.

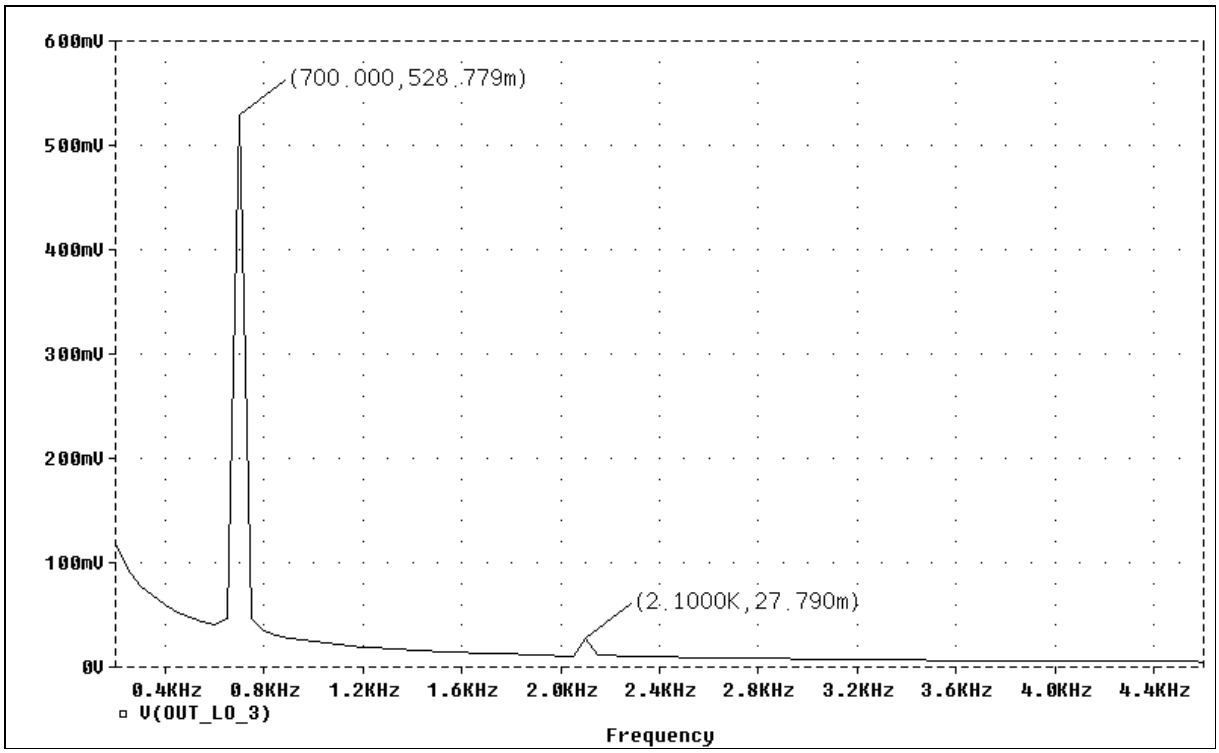

Figure 6: Amplitude Spectrum of a Square Wave Frequency of 697 Hz after passing through a 3rd order filter

In Figure 6, the amplitude spectrum of a square wave frequency of 697 Hz is shown, which has already passed through a 3rd order filter. The harmonics which result at 2091 Hz and 3485 Hz are sufficiently attenuated at -25.6 dB.

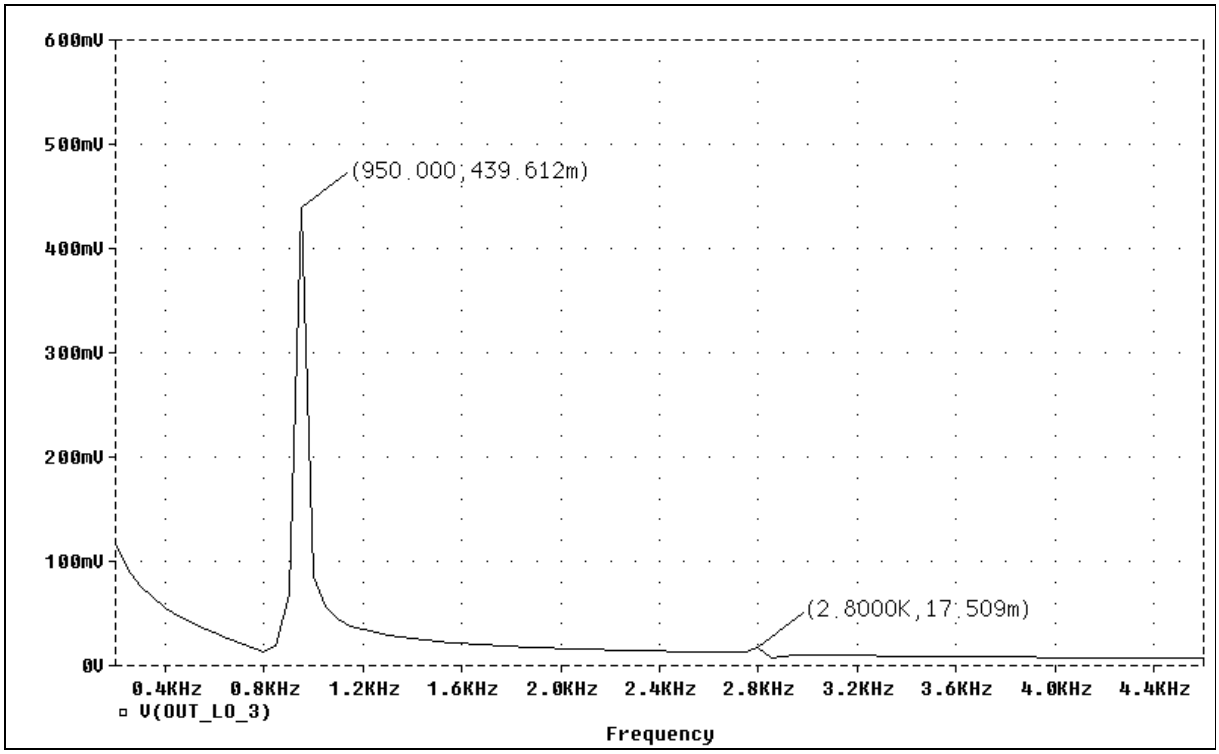

Figure 7: Amplitude Spectrum of a Square Wave Frequency of 941 Hz after passing through a 3rd order filter

Figure 7 shows a square wave frequency of 941 Hz. In the frequency range of interest up to 4600 Hz, only one harmonic is generated. After passing through the filter, this interference

frequency of 2823 Hz is well suppressed at -27.9 dB. The difference of level between the highest and the lowest group frequency amounts to 1.9 dB.

In order to be able to use low-cost components with wide tolerances in production, several additional simulation runs using components with specific tolerances were made. As a result of this, the necessary tolerance for resistors and capacitors was determined to be 10%.

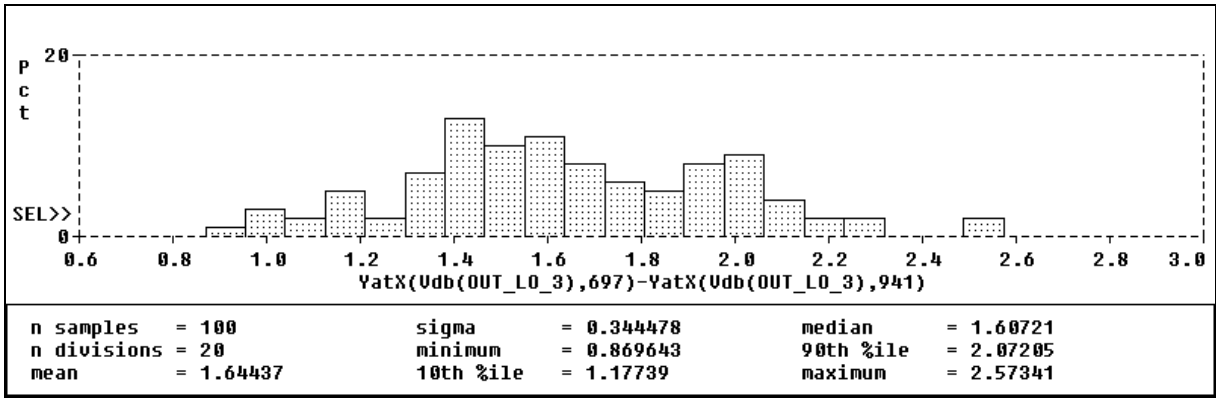

Figure 8: Histogram - Difference of Level within a Group

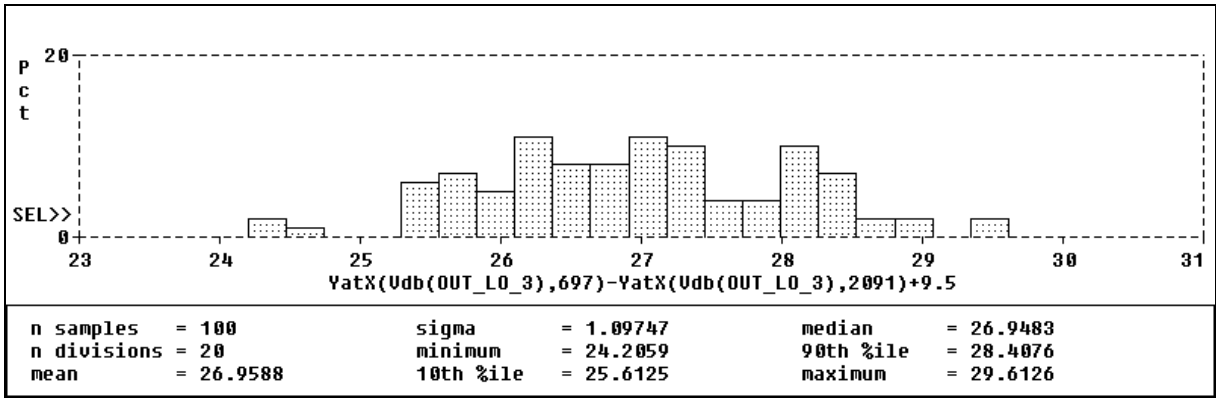

Figure 9: Histogram - Suppression of the Harmonics

Figures 8 and 9 show the histograms using a Monte Carlo analysis. In this case, the values of the components were changed randomly within a tolerance range of 10%. After 100 simulation runs, the results for all simulated filters were entered into a histogram. In the histogram in Figure 8, the difference of level within a group is shown. The maximum permissible difference of 3 dB between the highest and lowest group frequency is in no case exceeded. The average value of 1.6 dB is somewhat below the design goal of 2 dB.

The attenuation of the harmonics of the lowest group frequency is shown in Figure 9. The specified attenuation of the harmonics of 20 dB is met very well in every case, with an average value of almost 27 dB. In the worst case, the first harmonic is suppressed by at least 24.2 dB.

The values calculated at the filter of the Lo-Group frequencies conform also to the values of the Hi-Group Filter.

The two filters are constructed in the same way. They differ only in the cutoff frequencies for the lower and upper frequency groups. R1 and C1 form a 1st order low-pass. Since the input resistance of the circuit is also affected by R1, the value of this component should not be chosen to be too low; otherwise the driving power of the microcontroller will be exceeded,

and the square waveform of the signal will be lost. In this case frequencies may be added to the signal, which being interference signals will have a negative effect on the signal-to-noise ratio.

The transfer functions of higher order filters can no longer be represented with passive filters. The 2nd order filter must therefore be constructed with an operational amplifier. The amplification of the active filter is fixed at 0.2 by means of R1-1 and R1-2. The signal will thus be somewhat attenuated. This is necessary in order not to overdrive the operational amplifier, since the peak value of the (sine wave) fundemental of the square-wave signal is more than the amplitude of the square-wave (see also Fourier Series and Figure 2). In the subsequent adding stage, the levels of the signals can anyway be adjusted. The direct component contained in the square-wave signal sets the operating point of the operational amplifier to Vcc/2 (see also Fourier series and Figure 2). This direct component signal must however not be attenuated by the input voltage divider R1-1/R1-2. The capacitor C3 therefore blocks this path from direct voltage.

The analog filters deliver at each of their outputs a sine-wave signal of the upper and of the lower frequency groups respectively. In the subsequent adder, these signals are added together. At this point the share of the upper and lower frequencies in the total signal, and thus the level of the output signal, can be adjusted with the two resistors R4 and R5. In this way, the output power can be very easily be adjusted to conform to the various specifications applying in different countries.

When calculating component values, as is usual the capacitors were pre-determined, and the values of resistors then calculated. For the actual construction, capacitors and resistors from the standard range E12 with 10% tolerances were used.

Figure 10 shows the circuit diagram of the analog filter with the subsequent adder:

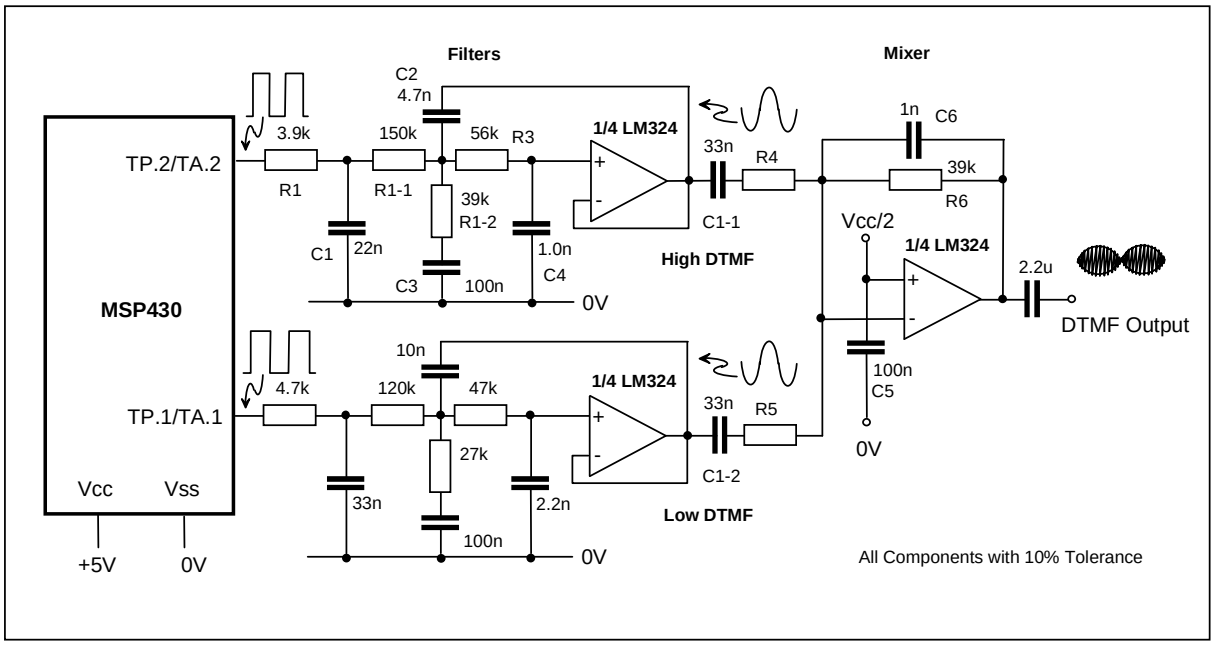

Figure 10: Circuit of Analog Filter followed by Adder circuit

The capacitors C1-1 and C1-2 couple the signal together at the working point of Vcc/2. Too high values should not be chosen for them, since these capacitors act as a high-pass filter to filter out low frequency interference. The blocking capacitor C5 filters out noise in the reference voltage. If an additional capacitor C6 is placed in parallel with the feedback coupling resistor R6, then a 1st order low-pass filter will be created. If a lower cutoff frequency is chosen, the additional filtering out of high frequency interference signals improves further the dynamic behavior of the output signal, but the high frequencies of the upper frequency group will be somewhat attenuated. If however in certain applications the generation of the highest DTMF frequency of 1633 Hz can be dispensed with, since it serves only to create the special characters A-D, then the signal-to-noise ratio can be improved by using a lower cutoff frequency for the filter. It is true that with higher cutoff frequencies the interference signal level will increase somewhat; however, the high frequencies of the DTMF signal will not be affected.

4 Measured values of the DTMF Transmitter

The spectrograms which follow (Figures 11 and 12) show the output signal of the DTMF transmitter at various DTMF frequencies. The amplitude spectrum of the character “1” is shown in Figure 11. The wanted signal frequencies at 697 Hz and 1207 Hz are at levels of -10.5 dB and -8.5 dB. The harmonics at 2091 Hz and 3621 Hz are attenuated with almost 30 dB. In representing the character “D”, two of the highest frequencies - namely 941 Hz and 1633 Hz - are generated. As shown in Figure 12, the level at the lower frequency is -12 dB and at the higher frequency is -11 dB. The corresponding harmonics are attenuated at more than 30 dB. The measured values thus confirm the results of simulations and conformance to the specification [1].

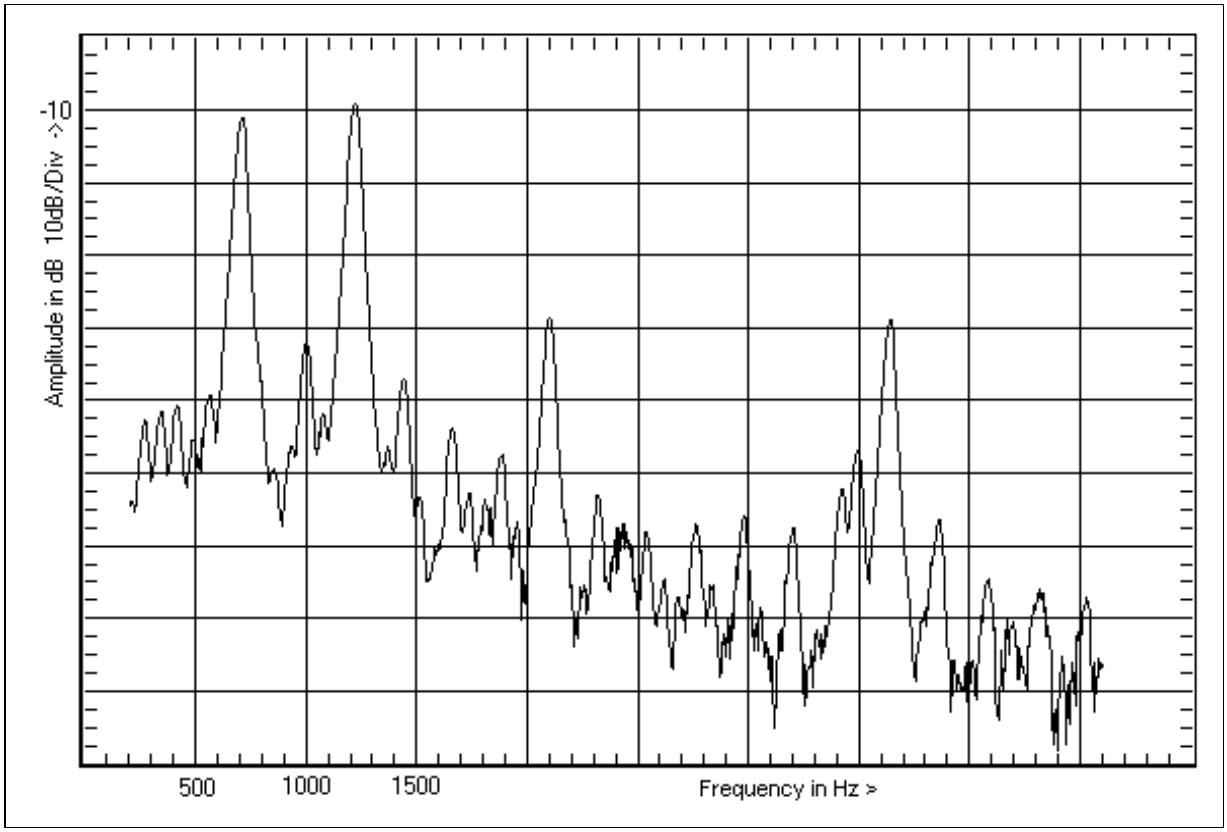

Figure 11: Amplitude Spectrum of the Character “1”: 697 and 1207 Hz

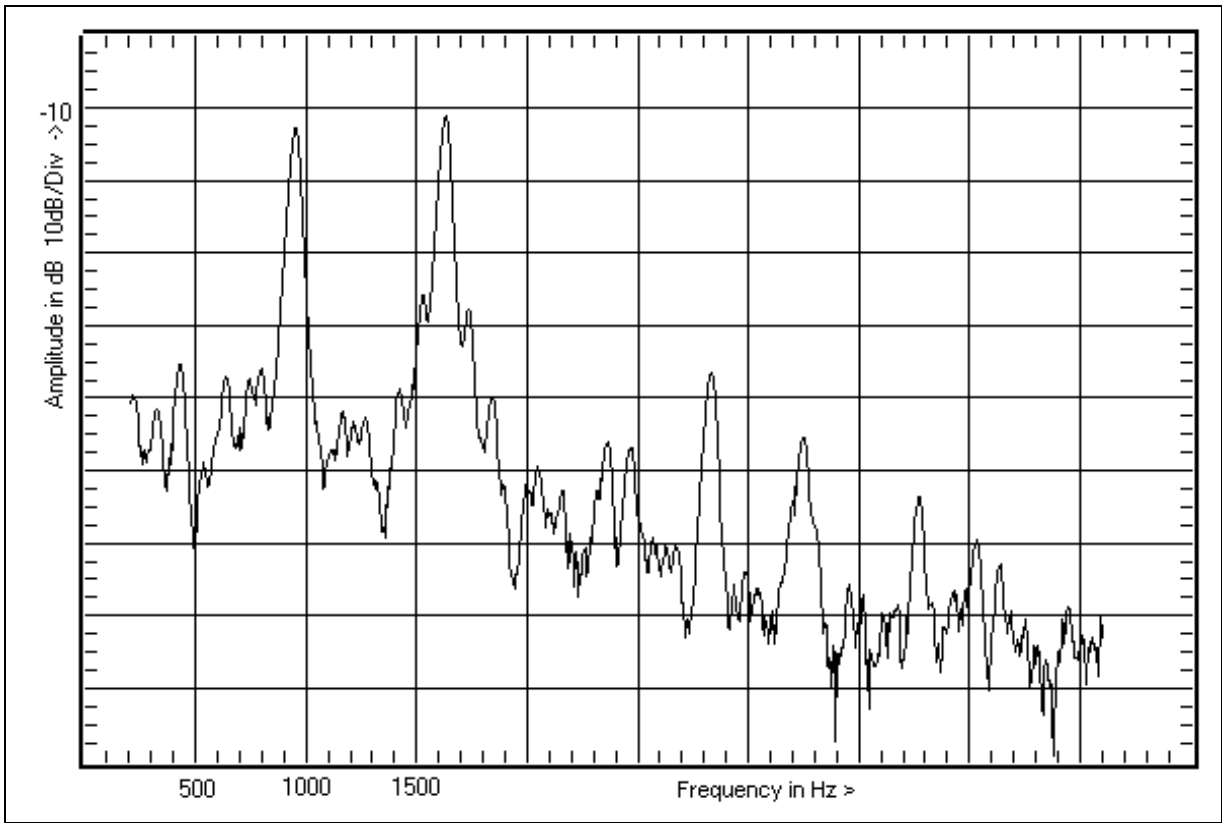

Figure 12: Amplitude Spectrum of the Character "D": 941 and 1633 Hz

The precision of the generated frequencies can not be given explicitly if the sqare-wave signals are generated with two different timers, because it depends on the combination of two frequencies and the timers which are used. This is a result of the collision of the timer interrupts. The specified tolerance of ±1.8 % is however very well fulfilled.

If an 8 Bit timer and the Timer Port timer are used with a MCLK of 1.048 MHz, then frequencies for the Lo-Group will be generated with an accuracy of better than 0.3 %. At the frequencies of the higher group, variations of not more than 0.5 % can in practice be attained. The only exception to this is the combination of the DTMF character "D". For this, the two highest frequencies must be generated. As a result, with this frequency combination for generating the Hi-Group frequency of 1633 Hz, a variation of -0.97 % must be tolerated. With the exception of this specific case, even the highest frequency of 1633 Hz is generated with an accuracy of better than 0.5 %. The maximum deviations are summarized again in the following table.

Lo-Group		Hi-Group	
Frequency in Hz	Maximum Deviation	Frequency in Hz	Maximum Deviation
697	-0.28 %	1207	+0.33 %
770	-0.13 %	1336	+0.45 %
853	±0.12 %	1477	±0.14 %
941	-0.21 %	1633	-0.97 %

If the Timer_A is used for the generation of the frequencies, then the error vary with the used MCLK:

MCLK MHz	1.048	2.096	3.144	3.800
FLL Multiplier N	32	64	96	116
697 Hz	+0.027%	+0.027%	+0.027%	+0.027%
770 Hz	-0.015%	-0.016%	+0.033%	-0.016%
852 Hz	+0.059%	-0.023%	+0.005%	+0.031%
941 Hz	+0.029%	+0.029%	+0.029%	+0.035%
1209 Hz	-0.079%	+0.036%	+0.036%	-0.003%
1336 Hz	+0.109%	-0.018%	+0.025%	+0.025%
1477 Hz	-0.009%	-0.009%	-0.009%	-0.009%
1633 Hz	+0.018%	+0.018%	+0.018%	+0.018%

5 Summary

The software for this application has been kept simple, and with about 300 Bytes needs little memory capacity in the RAM and ROM. As a result of the built-in timer module, the required frequencies can be generated very accurately without placing an undue load on the CPU. If the combination of an 8-Bit timer and Timer/Port timer is used for the generation of the frequencies, then the interrupt routines will place a load on the CPU of about 12%. If the frequencies can be generated with the Timer_A, then the loading on the CPU as a result of the interrupt routines will be reduced to 6%. As a result, during the DTMF transmission other tasks can be implemented, or the CPU can be switched into the Low-Power mode in order to save current.

The functionality of the module described for generating DTMF signals from sqare-wave signals has been demonstrated by constructing the necessary hardware. Since it was possible to use components with wide tolerances, the costs involved for this solution have been kept very low. The required specification has also been fulfilled very well, such that special DTMF transmitter modules are no longer needed in applications in which the MSP430 is used as the controller.

If in some applications it is necessary to improve the signal-to-noise ratio, an additional filter can be constructed with a fourth operational amplifier in order to suppress interfering frequencies still further. This additional operational amplifier is in any case available in a DIL14 package.

6 References

[1] *Bundesamt für Post und Telekommunikation* (Federal Office for Post and Telecommunications): BAPT 223 ZV 5, *Zulassungsvorschrift für Endeinrichtungen zur Anschaltung an analoge Wählanschlüsse (ausgenommen Notruf- und Durchwahlanschlüsse) des Telefonnetzes* (approval specification for end equipment to be connected to analog dialling connections of the telephone network, except for emergency and in-dialling connections) / ISDN of the *Deutschen Bundespost Telekom; Bundesministerium für Post und Telekommunikation*, Draft, Bonn April 1994

[2] Papula: *Mathematik für Ingenieure* 2 (Mathematics for Engineers); Vieweg Verlag, Braunschweig 1990

[3] Tietze / Schenk: *Halbleiterschaltungstechnik*; (Semiconductor Circuit Design), 10th. Edition; Springer Verlag, Berlin 1993

[4] Lutz Bierl / Texas Instruments: MSP430 Family, Metering Application Report, Texas Instruments, Issue 2.1, Jan 1997, SLAAE10B

[5] Texas Instruments: MSP430 Family, Architecture User's Guide and Module Library, Texas Instruments, 1996, SLAUE10B

[6] Texas Instruments: MSP430 Family, Software User's Guide, Texas Instruments, 1996

[7] Texas Instruments: MSP430 Family, Assembly Language Tools User's Guide, Texas Instruments, 1996

[8] Siwy, Robert: *Systementwicklung einer Telekom-Applikation zum Senden und Empfangen von DTMF-Signalen mit dem Microcontroller MSP430* (System development of a telecom application to send and receive digital signals with the Microcontroller MSP430); *Diplomarbeit, Fachhochschule* Landshut, Mai 1997

1 Introduction

The second part of the Application Report describes the reception and the recognition of DTMF signals using the Microcontroller MSP430. The theoretical and mathematical processes for designing digital filters are discussed. At the same time, these processes explain how suitable digital filters can be used to filter out specific frequency components from analog inputs. The AD converter in the configuration C32x is used for the digital conversion of the analog signal. 8 Boost wave digital filters are then computed in real time, and from their output values the DTMF character which has been received is recognized. This part includes a circuit diagram showing the connection of analog signals to the MSP430.

2 The Reception of DTMF Signals by means of Wave Digital Filters

In order to receive DTMF signals with a Microcontroller without the use of costly special components, signals which are received need to be processed with an Analog-to-Digital Converter (ADC), and recognized with a digital fíltering algorithm. The MSP430 is suitable for this purpose for the following reasons:

- Despite a relatively low clock frequency (up to 3.3 MHz), high computing speed is achieved as a result of commands which are processed within a clock cycle, as used in RISC computing.

- The connection of the controller to analog systems is very simple, because of the built-in ADC. External ADCs are not needed.

3 Basics of Digital Filtering

In analog circuit technology, active and passive filters are used for signal filtering; these consist of resistors, capacitors, inductors and amplifiers. The signals which have been processed are made up of voltages having waveforms which are continuous in time.

If digital signal processing is used instead, then the advantages of improved precision and reproducibility are obtained, such as can not be achieved with analog circuits as a result of the tolerance and aging of components. Digital filters consist of memories and computing circuitry. They are advantageous from the point of view of reducing the structural dimensions of integrated circuits. However, they also need analog-digital converters and digital-analog converters. Instead of processing continuously changing variables, digital filters process discrete sequences of digits.

These values are obtained by sampling a continuous signal at regular time intervals by means of an analog-digital converter; the ADC must contain a sample-and-hold function. This must conform to the sampling theorem [2], which says that the sampling frequency must be at least double that of the highest frequency occurring in the input signal. For this reason, an analog low-pass filter is placed before the ADC [3].

If an analog output is to be connected to the digital filter, then the computed values are again processed with a digital-analog converter, and finally smoothed with an analog low-pass filter [3].

Since in this application the intention was only an evaluation of the filtered signal, it was possible to dispense with this reconversion.

3.1 The Properties of Wave Digital Filters

The computation of digital filter algorithms usually requires very many multiplications. In order to reduce the magnitude of overflow and rounding errors, the word width must be made sufficiently wide. For this reason signal processors are preferable for this computation which are provided with a hardware multiplier. The processed bit widths are from 16 to 24 Bit. Some signal processors are also provided with floating-point calculation facilities, with which the data to be processed can be represented with floating-point precision.

In this case, Wave Digital Filters (WDFs) are used for the digital filtering. The WDFs {5, 107} are derived from analog LC and microwave filters. As a result, the particular properties of these analog filters are bestowed on the digital filters.

Wave Digital Filters have the special characteristic that they react with very little sensitivity to variations of the coefficients {5}. This is of particular importance in the application in question, because all multiplications must be calculated with the so-called Shift-and-Add process as a result of the lack of a multiplier. Shortening and optimization of the coefficients therefore saves calculation time. This simplification must however not have a negative effect on the stability or the frequency characteristics of the filter. As a result of shorter coefficients, the rounding noise resulting from multiplication is advantageously reduced. In addition, WDFs feature an excellent dynamic range {12, 22}.

WDFs remain stable should the input signal be suddenly discontinued. In consequence, it is possible to work through interrupt requests and subsequently to continue with the filter computation. Should the input of the filter be overloaded or there be other disturbances, the passive nature of the filter means that it recovers very quickly. This behavior is known as “Response Stability” {120}. For conventional IIR digital filters this problem has not been solved {5, S 302; 136}. WDFs even remain stable under loop conditions {19}.

3.2 The Structure of the Wave Digital Filter which is used

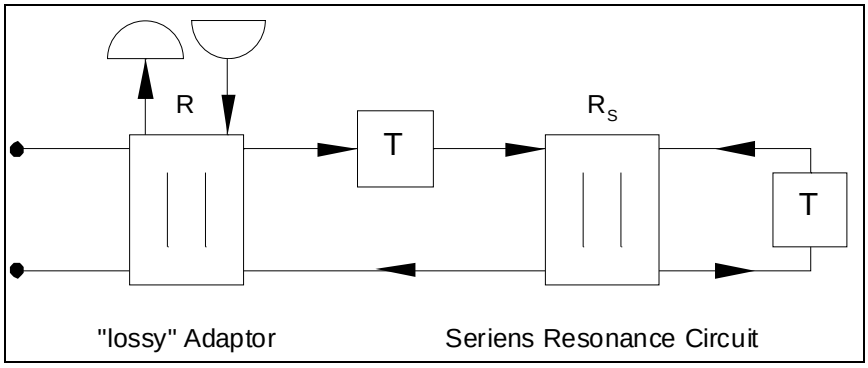

Figure 13: Structure of a Boost wave digital filter

For the task in question, a specific frequency must be recognized in the input signal. The Boost filter shown in Figure 13 is used for this purpose, consisting of a series resonance circuit and a “lossy” adapter [4]. The amplitude response of a filter of this kind is shown in Figure 14.

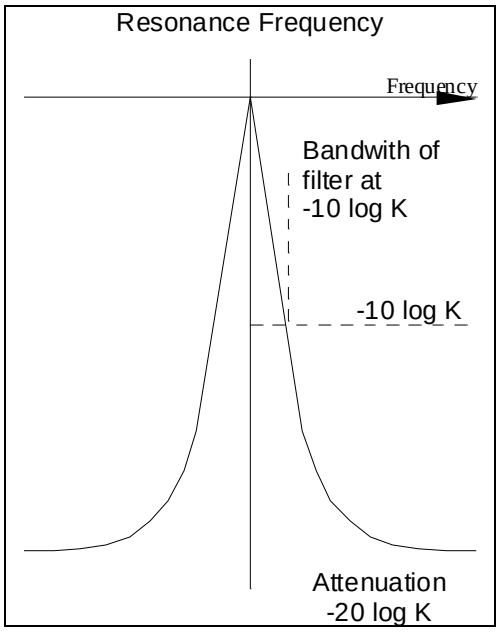

Figure 14: Amplitude response of a Boost wave digital filter

The filter is designed so that the resonance frequency is exactly the same as the frequency which is being sought. If the sought frequency occurs in the input signal, then it will be amplified whereas all other frequencies will be suppressed. The value of the amplification of the resonance frequency, or of the attenuation of all other frequencies, is specified with the parameter K, which is given in the unit -20 log K (dB). The bandwidth of the filter is given by determining the frequency at -10 log K (dB). If the output amplitude of the filter exceeds a certain value, then this indicates that the frequency being sought is present.

The adjustment of the amplification is performed with a lossy adapter. It has the same structure for all filters (see Figure 15). The series resonance circuit is composed of two delay elements and an adapter. Four different structures are possible, depending on the desired resonance frequency (Figure 16).

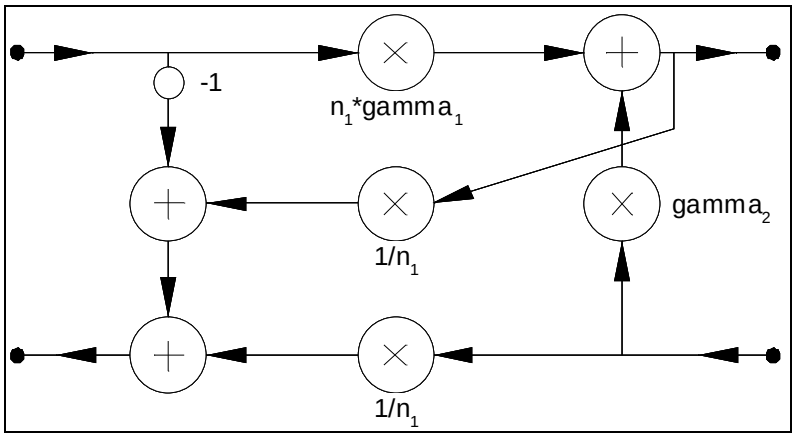

Figure 15: Structure of the "lossy" adapter

Since there is no floating-point arithmetic, the filter algorithm can be calculated to an integer accuracy of only 16 Bit; scaling factors are therefore incorporated in the adapters of types A and D, and this improves the dynamic performance of the filter.

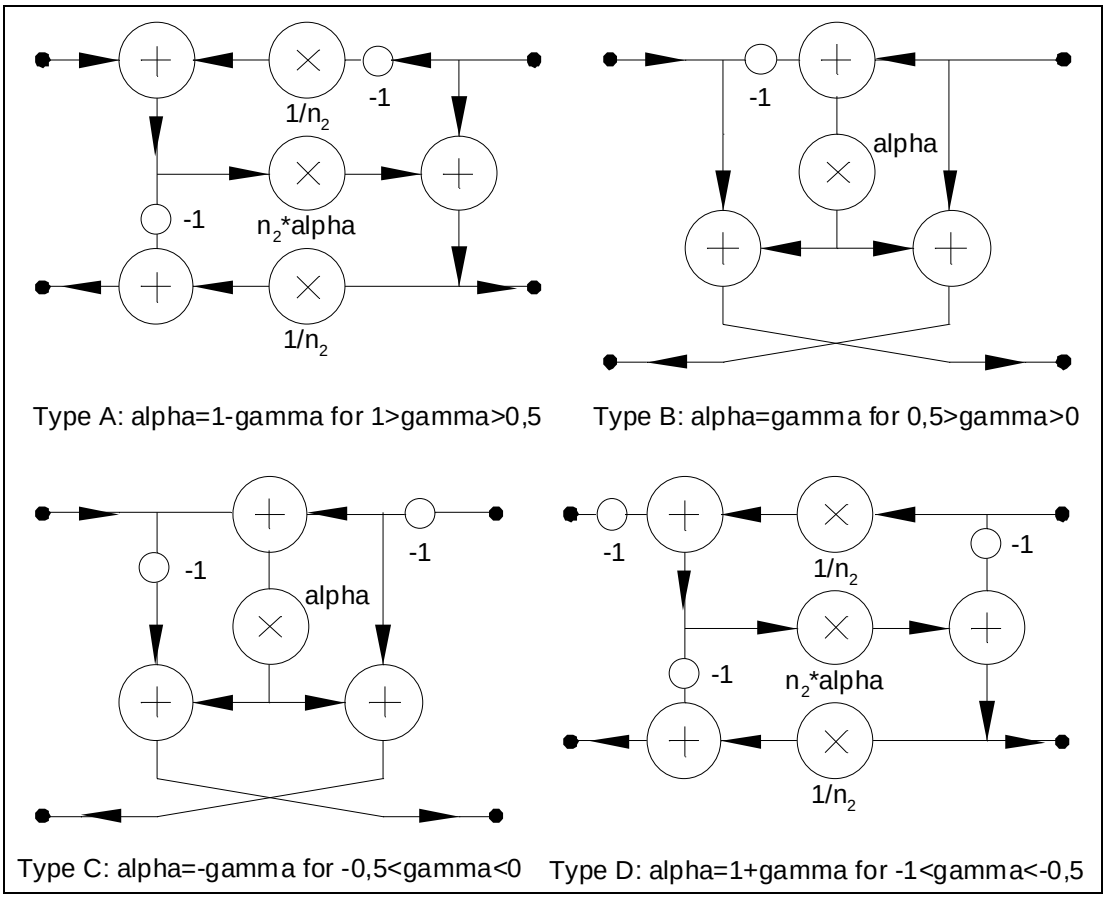

Figure 16: Four different possible configurations for two-port adapters

4 Representation of Numbers and Arithmetic

Since for reasons of cost the MSP430X325 contains no hardware multiplier, all multiplications must be performed with shifting and addition or subtraction operations. The multiplication of two dual numbers can be performed by repeated addition of the multiplicand. If a binary number is shifted n places to the right, the result is multiplication by 2^{-n}. Since a representation of the coefficients in 2^n intervals is usually insufficient, every '1' in the binary number must be replaced by means of a shift with subsequent addition. This will be explained with an example of the calculation of a multiplication according to the shift-and-add process:

1*0.46875	1b*0.011110b
$=2^{-2}$	=0.01b
$+2^{-3}$	+0.001b
$+2^{-4}$	+0.0001b
$+2^{-5}$	+0.00001b
=0.46875	=0.01111b

Since this number has very many places displaced from zero, many additions are needed to represent the number as a complement to two. In order to increase the processing speed in such cases, the Canonically Signed Digit Code (CSD) is used [5]. As a result of the use of the CSD Codes, the number of places of the coefficient displaced from zero are reduced to a minimum. This is done by inserting the value -1 in order to represent the value of the binary

number. This number is produced in the calculation by means of a subtraction. The above example using the CSD code then looks like this :

1*0.46875		1b*0.1000-10(CSD)
	$=2^{-2}$	=0.01b
	$\underline{-2^{-5}}$	<u>-0.00001b</u>
	=0.46875	=0.01111b

As a result of the use of the CSD code, the calculation of this multiplication has been reduced by two additions to one subtraction. The use of the CSD code is only useful with binary numbers which consist of a group of many ones.

Since the result of the multiplication of n Bits with m Bits is a product of n+m Bits, and the length of the word can not be increased from multiplication to multiplication, the product must be cut or rounded in the area of the less significant bits, if possible in such a way that the required calculation accuracy can be guaranteed. This is done with minimum errors using the Horner method [5].

In the following example, a 12 Bit product will be formed from a 12 Bit multiplicand and a 9 Bit multiplier.

The exact values are as follows:

X	*	Y		
0.11001011111b	*	0.01001001b		
0.796386719	*	0.28515625	=	0.22709465

The multiplication is performed with the Shift-and-Add process:

0.00110010111	$X*2^{-2}$		
0.00000110010	$X*2^{-5}$		
<u>0.00000000110</u>	$X*2^{-8}$		
0.00111001111		=	0.226074219 (-2.1 LSB)

The multiplication shows an error of more than 2 LSB, because a large number of low value bits are lost as a result of shifting to the right.

In contrast, the Horner method saves the low value Bits as long as possible:

0.00011001011	$X*2^{-3}$
<u>0.11001011111</u>	+ X
0.11100101010	
0.00011100101	$(X * 2^{-3} + X) * 2^{-3}$
<u>0.11001011111</u>	+ X
0.11101000100	
0.00111010001	$((X * 2^{-3} + X) * 2^{-3} X * 2^{-3} + X) * 2^{-2}$

= 0.227050781 (-0.1 LSB)

The process begins with the lowest value ‘1’ of the multiplier, followed by accumulation, after which the intermediate result is shifted to the right (by the difference of the bit positions until the next ‘1’), followed by addition and shifting etc. The absolute shift operations (in this case, 2^{-8}, 2^{-5}, 2^{-2}) are correctly implemented.

The above process is used for the implementation and optimization of the coefficients of the filter algorithm, in order to keep the computing requirements as low as possible whilst attaining the required accuracy .

Since the arithmetic processing in the microcontroller is usually made with 16 Bit wide integers, agreement must be reached on the representation of the signal values. Too much calculation time would be needed to implement Floating-Point arithmetic by means of software, and one is therefore obliged to use a Fix-Point system.

The following allocation of the 16 Bits applies for the filter algorithm described:

1 Bit as sign
2 Bit for the overflow part of calculations
Fixed point
13 Data Bits

The number 0.01111_B therefore appears in the internal representation as shown in Figure 17:

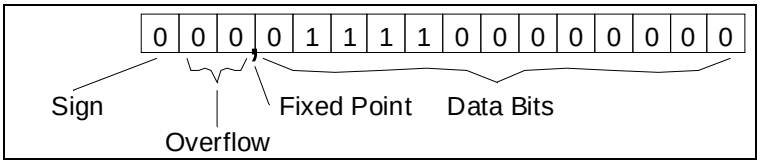

Figure 17: Digital Representation of the Signal Values

A negative coefficient is represented by splitting the coefficient into two parts. The first part is -1 and the second is the sum to give the negative coefficient.

Thus -0,3 is represented by -1 + 0,7.

5 Design of 8 Wave Digital Filters and Optimization of the Coefficients

Boost filters are used for frequency recognition, the calculation of which requires the following parameters [4]:

- Sampling frequency
- Resonance frequency of the filter
- Amplification (attenuation) of the filter in -20 log K (dB)
- Frequency at -10 log K (db)

Since the highest DTMF frequency is at 1633 Hz, the sampling theorem is fulfilled with a sampling frequency of 3640 Hz. This corresponds to a ninth of the ACLK of 32.768 kHz available in the MSP430, and for this reason can easily be generated with a timer. Although the speech signal taken to the ADC is then sampled at too low a rate, the specification of the

signal does not cause incorrect behavior in recognizing the DTMF, since the tone is only recognized as a DTMF signal when both DTMF frequencies are valid simultaneously for a certain period of time. In addition, during a DTMF transmission all other interference signals, and this means also speech, must be attenuated by 23 dB [1].

The resonance frequencies of the filters represent the individual DTMF frequencies. They should be taken without attenuation through the filters. The attenuation -20 log K (dB) of all other frequencies is fixed at -40dB. This corresponds to attenuation by a factor of K=100. If the frequency at -10 log K is applied to the adjacent DTMF frequency, then it will be ensured that the level of the frequency being sought is a factor of 10 larger than the adjacent frequency. This corresponds to a difference of 20 dB. As a result of the subsequent reduction and optimization of the coefficients, the factor of 10 can no longer be completely reached, but the remaining level margin is still completely adequate for reliable recognition of the frequency. If the output values of all filters are compared, then the application of a DTMF frequency to the filter whose resonance frequency coincides with the applied frequency results in a value 10 times greater than from the other filters.

Under the above conditions, the coefficients for the 8 filters are calculated as follows [4]:

Frequency in Hz		697	770	852	941	1209	1336	1477	1633
Reso-	Ad. type	B	B	B	C	C	D	D	D
nance	alpha	0.3594420	0.2393157	0.0999495	0.0534851	0.4935082	0.3291608	0.1702124	0.0516459
circuit	n2	--	--	--	--	--	1.5146077	2.4736004	5.8005043
“Lossy”	gamma 1	0.9773007	0.9774177	0.9759909	0.9749654	0.9696236	0.9702183	0.9692265	0.9694046
adapter	gamma 2	-0.9743449	-0.9745813	-0.9716988	-0.9696270	-0.9588356	-0.9600369	-0.9580333	-0.9583930
	n1	0.0250727	0.0248447	0.0276217	0.0296126	0.0399140	0.0387729	0.0406752	0.0403340

Since these numbers can only with difficulty be represented in a binary system and their calculation would require very many calculation steps, in the next step the coefficients must be optimized and shortened. The coefficients n1 and n2 are only scaling factors and can therefore be chosen relatively freely. Because of this they are rounded down to the next smaller power of two, this then reducing their calculation to a few shifting operations. The values for the alpha coefficients require the investment of more effort, because they influence the resonance frequency. In order to make a judgment of the frequency variation, the resonance frequency is again derived from the shortened coefficients. The following values for alpha are obtained after successful optimization:

Frequency	697	770	852	941	1209	1336	1477	1633
Choosen alpha(*n2)	0.359375	0.234375	0.09375	0.046875	0.5	0.328125	0.171875*2	0.046875*4
Resonance frequency	697	773	855	937	1213	1337	1475	1642
Variation in %	0.00	0.39	0.35	-0.43	0.33	0.07	-0.14	0.55
alpha(*n2) in CSD representation	0.010111	0.010000-1	0.00011	0.000011	0.1	0.010101	0.01011	0.0011

Despite in some cases a considerable shortening of the values of numbers and the resulting calculation time, only a slight variation from the nominal frequency needs to be tolerated.

Closer examination of the coefficients of the “lossy” adapter reveals their similarity. This is understandable, because these values are primarily determined by the parameter -10 log K, which is decided for all filters according to the same criteria: the adjacent DTMF frequency should be attenuated with -10 log K .

As a result, a single “lossy” adapter for all 8 DTMF filters is derived from the rounded values. This brings many advantages:

- Only one adapter, instead of 8, needs to be programmed.
- The optimization of the assembler codes is simpler with one adapter.
- If the same program parts are used, trouble shooting is considerably simplified.
- If sufficient calculation time is available, memory can be saved by calling a subroutine.

The optimized dual numbers of the values can be seen in the following table:

Frequency range	697 Hz - 1633 Hz
n1*gamma 1	0,00001b
gamma 2	-1,0 + 0,00001b
1/n1	32

6 Verification of the Filter Designs with a mathematical Simulation Program

The ADC built into the MSP430 provides a bridge from the analog to the digital world. However, in order to examine the characteristics of the filters, the exact values at the filter outputs are needed. Since it is not possible to get an output of the calculated value (there is no built-in DA converter), it is not possible to verify the amplitude characteristics of the filter. In order to obtain even so information about the filter characteristics which have been achieved, the algorithm is simulated with a mathematical program and the amplitude characteristics calculated.

In contrast to microcontrollers which calculate with 16 Bit precision, the algorithm in this case operates with floating point accuracy. The shortened coefficients are however used. As an example, a simulation program and the resulting amplitude characteristic of the 770 Hz filter is shown. For comparison, the amplitude characteristic with unrounded coefficients is also shown dotted (Figure 18):

```
f(alpha, gamma_1_n1, gamma_2, n1) :=
    t1 ← 0
    t2 ← 0
    for i ∈ 0..N
        k1 ← (t2 − t1)·alpha
        k3 ← k1 + t2
        k2 ← k1 + t1
        t2 ← k2
        t1 ← k3·gamma_2 + x_i·gamma_1_n1
        y_i ← (k3 + t1)/n1 − x_i
    y
```

u := f(0.234375 , 0.0242837 , - 0.9745813 , 0.0248447)

Function with unrounded values of gamma_1_n1, gamma_2, and n1

$$z := f\left[0.234375, \frac{1}{32}, \left(\frac{1}{32} - 1\right), \frac{1}{32}\right]$$

Function with rounded values

Program for the verification of the 770 Hz filter with a mathematical simulation program

- The resonance frequency lies at 773 Hz, and therefore the precalculated value for alpha=0.234375 is reached exactly.

- The amplification of the filter is only -36 dB instead of -40 dB. This is a result of the rounded coefficients gamma_1 and gamma_2. If the simulation is done with the exact values, then the desired amplification of -40 dB will be obtained.

- The adjacent DTMF frequency (697 Hz) is amplified with -18.5 dB, corresponding to an eighth of the input value. This separation is sufficient for unambiguous recognition of the frequency.

This makes clear one of the excellent properties of Wave Digital Filters, namely their insensitivity to rounding of the coefficients: despite considerable shortening of the coefficients, and consequently of the calculation time, the characteristics of the filter are maintained.

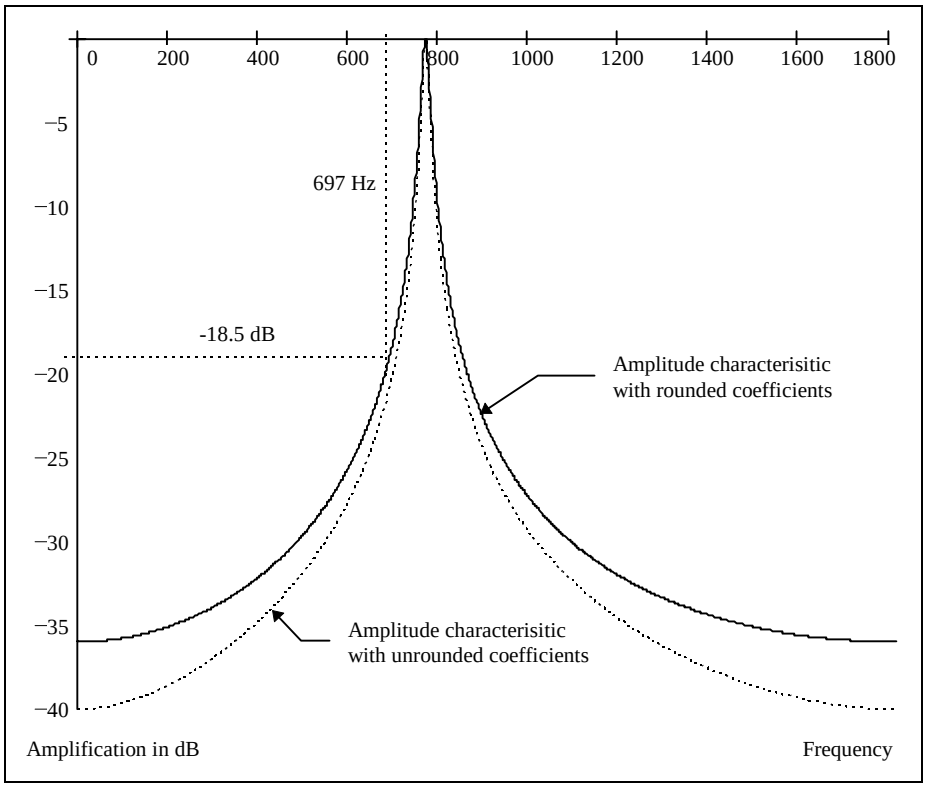

Figure 18: Amplitude characteristic of the 770 Hz filter simulated with MathCad

7 Software for Digital Filter Algorithms

Digital filter algorithms need a lot of computing power, because many multiplications and additions are needed for their calculation - particularly when the multiplications must be performed without hardware multipliers, and calculated with shift and adding operations. As a result, digital signal processors are often used for this, because they are provided with a fast hardware multiplier having an accumulator (MAC), or at least with a Barrel Shifter, this allowing shift operations in many places to be performed in a single process. It is therefore unusual to implement a filter algorithm in real time with a slow microcontroller.

This can be done even so with the microcontroller MSP430, because instructions for which the operands are in registers are performed within a single clock cycle. In addition, to increase the computing speed the system clock rate is raised from 1 MHz to 3.3 MHz.

The connection to analog systems is made simple by the built-in ADC. The high resolution of 14 Bit guarantees good dynamic performance in the subsequent digital filter. As a result, signals of low amplitude will be computed with the same accuracy as those with greater amplitude. It is therefore possible to dispense with an external ADC.

If algorithms need to be processed in a real time system, then the calculation of a piece of data must be completed before the next data arrives for calculation. The intermediate storage of data in a RAM is not possible, because at this point capacity is available for only a few words of data. The data is delivered from the ADC during a pre-determined time period.

An interrupt frequency of 3640 Hz has been chosen for the TP timer. This corresponds to nine ACLK periods. The ADC is started when the TP timer interrupt occurs. Since the ACLK is taken directly from the 32.768 kHz quartz crystal oscillator, a sample rate is obtained which is free of jittering and has quartz-crystal stability, and which so allows correct sampling of the

analog signal, and therefore reliable computed values [10]. When the conversion is finished, it is followed by the computation of 8 digital filters and the recognition of DTMF frequencies from the filter values. A time of 274.7 µs is therefore available for the calculation and recognition algorithm. This corresponds to 915 cycles at a cycle time of about 300 ns. After this time has elapsed, the computation must begin with the next A/D values. The process of the algorithm is shown in Figure 19.

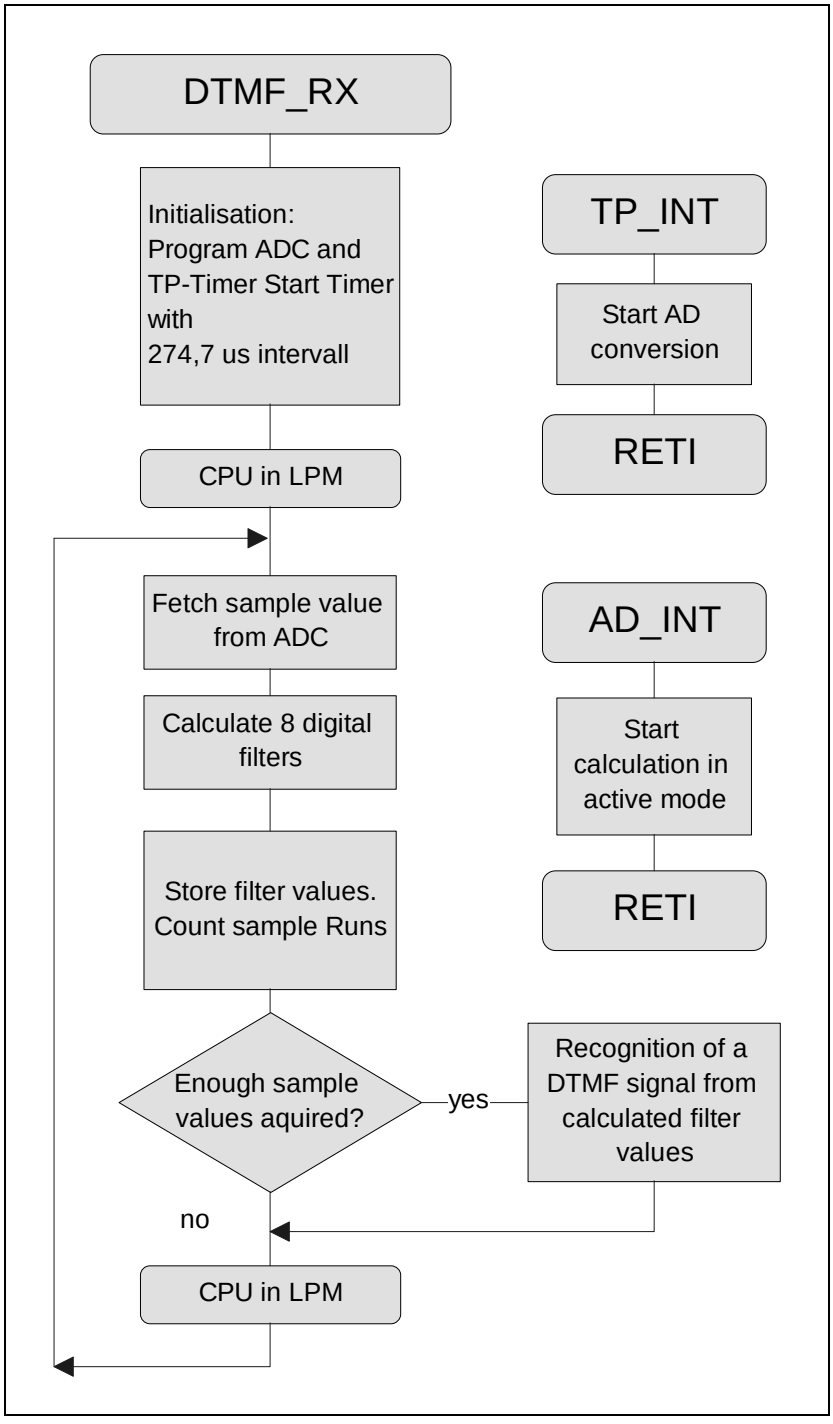

Figure 19: Algorithm for the real-time computation of 8 digital filters, and the recognition of DTMF Signals

8 Software for the Recognition of DTMF Signals

In order to decode the DTMF characters which have been received from the calculated filter values, a decoder algorithm must follow the computation of the filter. This must allocate two recognized frequencies to a DTMF character. In addition, a comparison of the filter values is performed during the filter calculation. In practice, for filters from the Hi-Group and Lo-Group separately, the filter with the highest output value is identified and the amplitude of the output value stored. Thus after each calculation of the 8 filters, the highest Lo-Group and Hi-Group frequencies of the last run which have occurred is obtained. Figure 20 shows the flow diagram of the decoder algorithm:

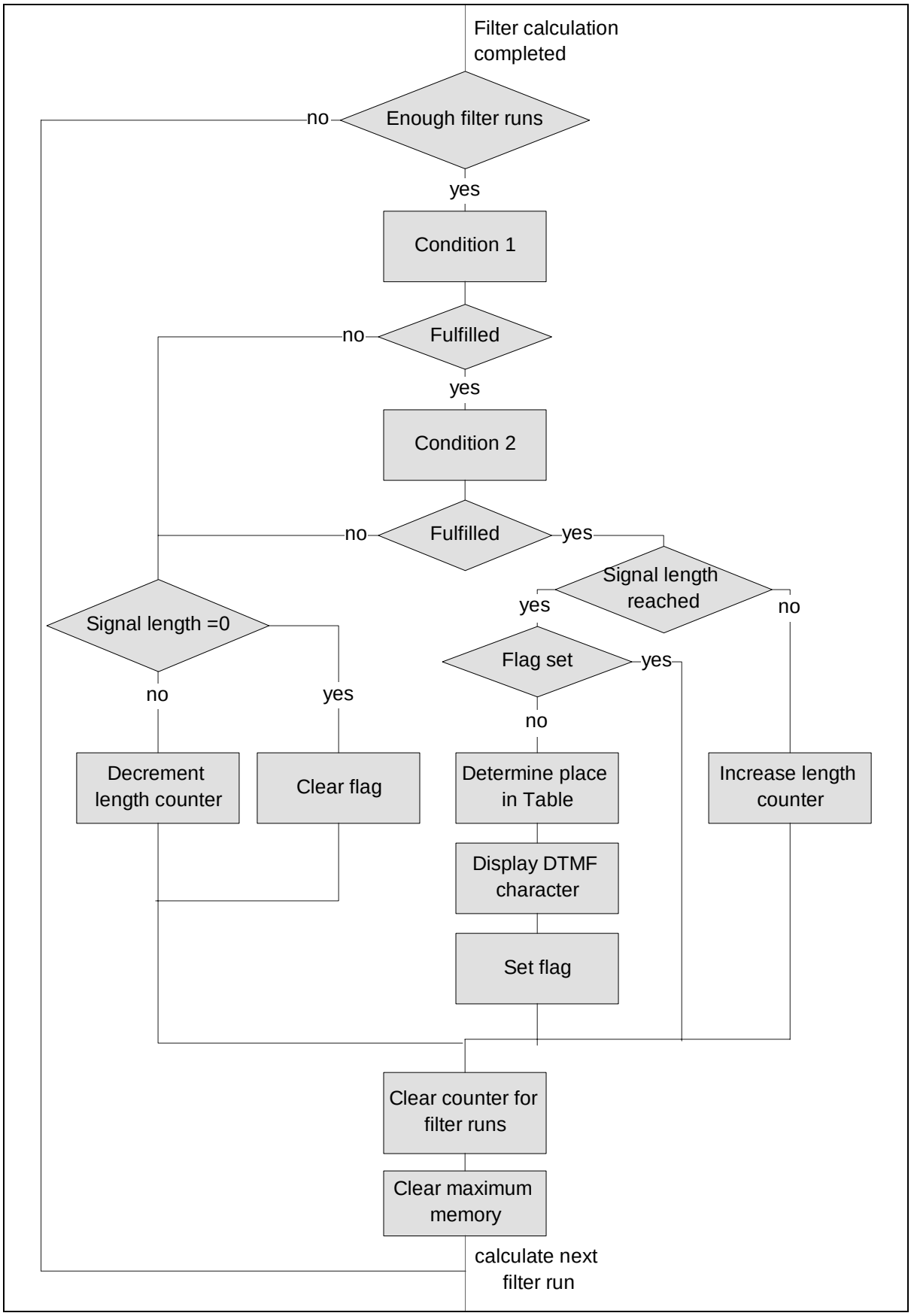

Figure 20: Flow diagram of the decoder algorithm

In order to ensure that all filters have reached a steady state, and the respective frequencies have gone through their maximum amplitudes, 20 filter passes are always consolidated. If enough samples have been calculated, the two maximum values from the Hi and Lo Group are tested for their validity under two conditions. The first condition checks whether both maximum values lie above the natural noise level. The second conditions ensures that the difference between the maximum values of both frequencies is not excessive.

These conditions can of course be extended further in order to improve the reception of particularly weak signals.

If both conditions have been fulfilled, then the two DTMF frequencies are valid. These frequencies must however only be recognized as DTMF signals after a specific time has elapsed. The signal duration is therefore recorded with a duration counter. If the specific signal duration has been reached, then the DTMF character is created from the frequencies in the row and column of a table. In order to avoid recognizing the same character a second time, should the signal duration be exceeded, a flag is set which displays the validity of the character which has been received.

If one of these conditions has not been fulfilled, then the duration counter is decremented. If the counter reaches zero, then this will be interpreted as a signal pause which separates several successive dialing characters from one another. The flag can now be cleared.

Figure 21 shows the timing conditions for the recognition of a DTMF signal in graphical form:

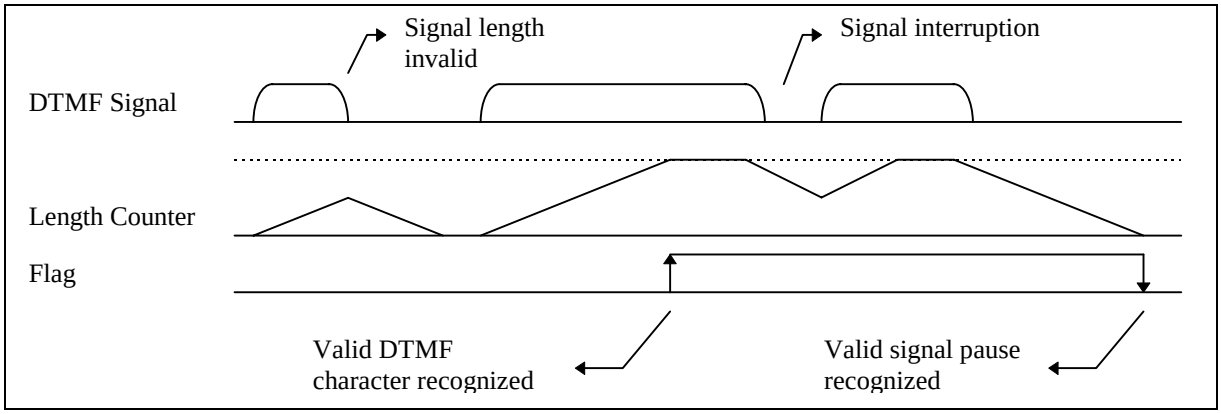

Figure 21: Timing diagram for the recognition of a DTMF signal

```
;    This Software calculates 8 Wave Digital Filters in order
;    to detect DTMF Signals.
;    The analog input Signal is sampled at pin A4 with a
;    samplerate of 3640 Hz
;    Recognized DTMF Characters are displayd on the LCD
;    Robert Siwy, October 1997

FILTER_1  .equ 1         ;enable 1633 Filter
FILTER_2  .equ 1         ;enable 1477 Filter
FILTER_3  .equ 1         ;enable 1336 Filter
FILTER_4  .equ 1         ;enable 1209 Filter
FILTER_5  .equ 1         ;enable 941 Filter
FILTER_6  .equ 1         ;enable 852 Filter
```

```
FILTER_7  .equ 1         ;enable 770 Filter
FILTER_8  .equ 1         ;enable 697 Filter

; USED HARDWARE DEFINITIONS
STACK     .equ      400h
TCCTL     .equ      42h
TCPLD     .equ      43h
TCDAT     .equ      44h
BTCTL     .equ      40h
LCDCTL    .equ      30h
IE1       .equ      00h
IE2       .equ      01h
SCFI      .equ      50h
SCFQCTL   .equ      52h
WDTCTL    .equ      0120h
WDTCL     .equ      88h
WDTPW     .equ      05a00h
ACTL      .equ      0114h
PD        .equ      1000h
ADAT      .equ      0118h

; SOFTWARE DEFINITIONS
DISPL     .equ      031h ;Base adress of LC-Display
LPM0      .equ      010h
LENGTH    .equ      7    ;(Length+1) * 5.48ms=Signallength
THRE      .equ      100  ;threshold for noise

FLAG_REG  .equ      r4
LCOUNT    .equ      r5
ROW       .equ      r6        ;Lo-Group Row
COL       .equ      r7        ;Hi-Group Column
MAXLO     .equ      r8
IN        .equ      r9        ;Input Register
MAXHI     .equ      r10
OUT       .equ      r13
COUNT     .equ      r15

;*****************************************************************
;          Memory Allocation for 8 Filter
;*****************************************************************
          .bss  T1_1633, 2
          .bss  T2_1633, 2
          .bss  T1_1477, 2
          .bss  T2_1477, 2
          .bss  T1_1336, 2
          .bss  T2_1336, 2
          .bss  T1_1209, 2
          .bss  T2_1209, 2
          .bss  T1_941, 2
          .bss  T2_941, 2
          .bss  T1_852, 2
          .bss  T2_852, 2
          .bss  T1_770, 2
```

```
          .bss  T2_770, 2
          .bss  T1_697, 2
          .bss  T2_697, 2

;***************************************************************
;          Program Code
;***************************************************************
               .text
INIT
     mov       #STACK-2,SP          ;initialize Stack
     mov       #WDTPW+WDTCL,&WDTCTL     ;hold watchdog
     ;Init. MCLK
     mov.b     #102-1,&SCFQCTL      ;102*32.768Hz=3,342MHz
     bis.b     #008h,&SCFI0         ;Freq. Integr. to 3MHz
     ;Init. LCD/clear LCD Memory
CLRSCR
     mov       #11,r5
clr1
     clr.b     DISPL-1(r5)
     dec       r5
     jnz       clr1

     mov.b     #017h,&BTCTL   ;LCD-Timing
     mov.b     #0ffh,&LCDCTL  ;LCD-Gen.
     ;Init. AD-Wandler
     mov       #04912h,&ACTL  ;Init. ADC with internal
                             ;Reference
     bis.b     #004h,&IE2     ;enable ADC-Int.
     ;Init. 8B-Timer
     mov.b     #100h-9,&TCPLD ;8B-Timer to 9 ACLK=274us
     mov.b     #000h,&TCDAT   ;load Counter
     mov.b     #068h,&TCCTL   ;Init. 8B-Timer
     mov.b     #008h,&IE1     ;enable 8B-Timer Int.
     eint
     bis.b     #LPM0,SR       ;CPU off

DTMF_FILTER
     inc       COUNT
     cmp       #20,COUNT      ;20 sample values?
     jlo       FILTER         ;not 20 Samples, jump
     ;condition 1
     cmp       #THRE,MAXLO    ;Maximum from Lo below threshold?
     jlo       FALSE          ;yes, jump out

     cmp       #THRE,MAXHI    ;Maximum from Hi below threshold?
     jlo       FALSE          ;yes, jump out
     ;condition 2
     rra       MAXLO          ;divide MAXLO by two
     cmp       MAXLO,MAXHI    ;MAXHI > MAXLO/2
     jlo       FALSE          ;yes, jump out

     rla       MAXLO          ;restore MAXLO
     rra       MAXHI          ;divide MAXHI by two
     cmp       MAXHI,MAXLO    ;MAXLO > MAXHI/2
```

```
     jlo       FALSE          ;yes, jump out

     cmp       #LENGTH,LCOUNT ;Signal length valid?
     jeq       DISPLAY        ;yes, jump to display
     inc       LCOUNT         ;no, increase length counter
     jmp       CONTINUE
DISPLAY
     bit       #01h,FLAG_REG  ;test Flag
     jnz       CONTINUE       ;already recognized, continue
     add       ROW,COL        ;add ROW and COLUMN for table
     mov.b     DTMF_Tab(COL),DISPL ;display DTMF Character
     bis       #01h,FLAG_REG  ;set Flag
     jmp       CONTINUE
FALSE
     tst       LCOUNT
     jnz       FALSE1
     mov.b     #008h,DISPL+3  ;display valid pause
     mov.b     #008h,DISPL+4  ;    "    "
     bic       #01h,FLAG_REG  ;clear Flag
     jmp       CONTINUE
FALSE1
     dec       LCOUNT

CONTINUE
     clr       MAXHI
     clr       MAXLO
     clr       COUNT

FILTER
     mov   &ADAT,IN      ;Sample to r9/IN
     sub   #01fffh,IN    ;form signed value
     rla   IN            ;*2
     rla   IN            ;*2

     .if   FILTER_1
FILTER_1633
     mov  T2_1633,r12    ;N11=T2/4-T1
     mov  r12,r13
     rra  r12
     rra  r12
     sub  T1_1633,r12    ;N11 in r12

     mov  r12,r14        ;N11*2*alpha-T2
     rra  r12
     rra  r12
     rra  r12
     mov  r12,r11
     rra  r12
     add  r12,r11
     sub  r13,r11        ;N15 in r11
     mov  r11,T2_1633    ;N3=N15/4-N11
     rra  r11
     rra  r11
```

```
     sub  r14,r11        ;N3 in r11
          ;"Common" Lossy-Adaptor
          ;expects N3 in r11
          ;1/n1=32, gamma_2=-1+1/32)
          ;n1*gamma_1=1/32
     mov  r11,r13        ;save N3 to r13
     mov  IN,r12         ;T1=N3*gamma_2+n1*gamma_1*IN
     rra  r12
     rra  r12
     rra  r12
     rra  r12
     rra  r12            ;n1*gamma_1*IN in r12
     sub  r11,r12
     rra  r11
     rra  r11
     rra  r11
     rra  r11
     rra  r11
     add  r11,r12        ;T1 in r12
     mov  r12,T1_1633    ;save T1 to RAM

     rla  r13            ;N3 * 32
     rla  r13
     rla  r13
     rla  r13
     rla  r13

     rla  r12            ;T1 * 32
     rla  r12
     rla  r12
     rla  r12
     rla  r12
     sub  IN,r12
     add  r12,OUT        ;r13 is OUT
          ;***********************************
     cmp  OUT,MAXHI      ;compare Output with MAX
     jge  SWAP1
     mov  OUT,MAXHI      ;big value to MAX
     mov  #03,COL        ;set Column to 3
SWAP1     ;***********************************
          ;Standard-Lossy End
FILTER_1633_END
          .endif

          .if   FILTER_2
FILTER_1477
     mov  T2_1477,r12    ;N11=T/2-T1
     mov  r12,r13
     rra  r12
     sub  T1_1477,r12    ;N11 in r12

     mov  r12,r14        ;N15=N11*2*alpha-T2
     rra  r12
     rra  r12
```

```
     mov  r12,r11
     rra  r12
     rra  r12
     add  r12,r11
     rra  r12
     add  r12,r11
     sub  r13,r11        ;N15 in r11
     mov  r11,T2_1477    ;N3=N15/2-N11
     rra  r11
     sub  r14,r11        ;N3 in r11
          ;"Common" Lossy-Adaptor
          ;expects N3 in r11
          ;1/n1=32, gamma_2=-1+1/32)
          ;n1*gamma_1=1/32
     mov  r11,r13        ;save N3 to r13
     mov  IN,r12         ;T1=N3*gamma_2+n1*gamma_1*IN
     rra  r12
     rra  r12
     rra  r12
     rra  r12
     rra  r12            ;n1*gamma_1*IN in r12
     sub  r11,r12
     rra  r11
     rra  r11
     rra  r11
     rra  r11
     rra  r11
     add  r11,r12        ;T1 in r12
     mov  r12,T1_1477    ;save T1 to RAM

     rla  r13            ;N3 * 32
     rla  r13
     rla  r13
     rla  r13
     rla  r13

     rla  r12            ;T1 * 32
     rla  r12
     rla  r12
     rla  r12
     rla  r12
     sub  IN,r12
     add  r12,OUT        ;r13 is OUT
          ;***********************************
     cmp  OUT,MAXHI      ;compare Output with MAX
     jge  SWAP2
     mov  OUT,MAXHI      ;big value to MAX
     mov  #02,COL        ;set Column to 2
SWAP2
          ;***********************************
          ;Common-Lossy End
FILTER_1477_ENDE
          .endif
```

```
          .if  FILTER_3
FILTER_1336
     mov   T2_1336,r12   ;N11=T2-T1
     mov   r12,r13
     sub   T1_1336,r12   ;N11 in r12

     mov   r12,r14       ;N15=N11*alpha-T2
     rra   r12
     rra   r12
     mov   r12,r11
     rra   r12
     rra   r12
     add   r12,r11
     rra   r12
     rra   r12
     add   r12,r11
     sub   r13,r11       ;N15 in r11
     mov   r11,T2_1336
     sub   r14,r11       ;N3=N15-N11 in r11
           ;"Common" Lossy-Adaptor
           ;expects N3 in r11
           ;1/n1=32, gamma_2=-1+1/32)
           ;n1*gamma_1=1/32
     mov   r11,r13       ;save N3 to r13
     mov   IN,r12        ;T1=N3*gamma_2+n1*gamma_1*IN
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12           ;n1*gamma_1*IN in r12
     sub   r11,r12
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     add   r11,r12       ;T1 in r12
     mov   r12,T1_1336   ;save T1 to RAM

     rla   r13           ;N3 * 32
     rla   r13
     rla   r13
     rla   r13
     rla   r13

     rla   r12           ;T1 * 32
     rla   r12
     rla   r12
     rla   r12
     rla   r12

     sub   IN,r12
     add   r12,OUT       ;r13 is OUT
```

```
            ;***********************************
     cmp    OUT,MAXHI      ;compare Output with MAX
     jge    SWAP3
     mov    r13,MAXHI      ;big value to MAX
     mov    #01,COL        ;set Column to 1
SWAP3       ;***********************************
            ;Common-Lossy End
FILTER_1336_ENDE
            .endif

            .if  FILTER_4
FILTER_1209
     mov    T2_1209,r13    ;N1=(T1-T2)*alpha
     mov    T1_1209,r11
     mov    r11,r14
     sub    r13,r11
     rra    r11            ;N1 in r11
     mov    r11,r12        ;N3=N1-T2
     sub    r13,r11        ;N3 in r11

     sub    r14,r12        ;N2=N1-T1 in r12
     mov    r12,T2_1209
            ;"Common" Lossy-Adaptor
            ;expects N3 in r11
            ;1/n1=32, gamma_2=-1+1/32)
            ;n1*gamma_1=1/32
     mov    r11,r13        ;save N3 to r13
     mov    IN,r12         ;T1=N3*gamma_2+n1*gamma_1*IN
     rra    r12
     rra    r12
     rra    r12
     rra    r12
     rra    r12            ;n1*gamma_1*IN in r12
     sub    r11,r12
     rra    r11
     rra    r11
     rra    r11
     rra    r11
     rra    r11
     add    r11,r12        ;T1 in r12
     mov    r12,T1_1209    ;save T1 to RAM

     rla    r13            ;N3 * 32
     rla    r13
     rla    r13
     rla    r13
     rla    r13

     rla    r12            ;T1 * 32
     rla    r12
     rla    r12
     rla    r12
     rla    r12
```

```
     sub   IN,r12
     add   r12,OUT       ;r13 is OUT
           ;***********************************
     cmp   OUT,MAXHI     ;compare Output with MAX
     jge   SWAP4
     mov   r13,MAXHI     ;big value to MAX
     mov   #00,COL       ;set Column to 0
SWAP4      ;***********************************
           ;Common-Lossy End
FILTER_1209_ENDE
           .endif

           .if  FILTER_5
FILTER_941
     mov   T2_941,r13    ;N1=(T1-T2)*alpha
     mov   T1_941,r12
     mov   r12,r14
     sub   r13,r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     mov   r12,r11
     rra   r12
     add   r12,r11
     mov   r11,r12       ;N1 in r11/r12
     sub   r13,r11       ;N3=N1-T2 in r11
     sub   r14,r12       ;N2=N1-T1
     mov   r12,T2_941
           ;"Common" Lossy-Adaptor
           ;expects N3 in r11
           ;1/n1=32, gamma_2=-1+1/32)
           ;n1*gamma_1=1/32
     mov   r11,r13       ;save N3 to r13
     mov   IN,r12        ;T1=N3*gamma_2+n1*gamma_1*IN
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12           ;n1*gamma_1*IN in r12
     sub   r11,r12
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     add   r11,r12       ;T1 in r12
     mov   r12,T1_941    ;save T1 to RAM

     rla   r13           ;N3 * 32
     rla   r13
     rla   r13
     rla   r13
```

```
     rla   r13

     rla   r12           ;T1 * 32
     rla   r12
     rla   r12
     rla   r12
     rla   r12

     sub   IN,r12
     add   r12,OUT       ;r13 is OUT
           ;***********************************
     cmp   OUT,MAXLO     ;compare Output with MAX
     jge   SWAP5
     mov   r13,MAXLO     ;big value to MAX
     mov   #012,ROW      ;set Row to 12
SWAP5      ;***********************************
           ;Common-Lossy End
FILTER_941_ENDE
           .endif

           .if  FILTER_6
FILTER_852
     mov   T2_852,r11    ;N1=(T2-T1)*alpha
     mov   r11,r12
     mov   T1_852,r13
     sub   r13,r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     mov   r12,r14
     rra   r12
     add   r12,r14       ;N1 in r14
     add   r14,r11       ;N3=T2+N1 in r11
     add   r14,r13       ;N2=T1+N1
     mov   r13,T2_852
           ;"Common" Lossy-Adaptor
           ;expects N3 in r11
           ;1/n1=32, gamma_2=-1+1/32)
           ;n1*gamma_1=1/32
     mov   r11,r13       ;save N3 to r13
     mov   IN,r12        ;T1=N3*gamma_2+n1*gamma_1*IN
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     rra   r12           ;n1*gamma_1*IN in r12
     sub   r11,r12
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     rra   r11
```

```
     add   r11,r12        ;T1 in r12
     mov   r12,T1_852     ;save T1 to RAM

     rla   r13            ;N3 * 32
     rla   r13
     rla   r13
     rla   r13
     rla   r13

     rla   r12            ;T1 * 32
     rla   r12
     rla   r12
     rla   r12
     rla   r12
     sub   IN,r12

     add   r12,OUT        ;r13 is OUT
           ;*************************************
     cmp   OUT,MAXLO      ;compare Output with MAX
     jge   SWAP6
     mov   r13,MAXLO      ;big value to MAX
     mov   #08,ROW        ;set Row to 8
SWAP6      ;*************************************
           ;Common-Lossy End
FILTER_852_ENDE
           .endif

           .if  FILTER_7
FILTER_770
     mov   T2_770,r11      ;N1=(T2-T1)*alpha
     mov   r11,r12
     mov   T1_770,r13
     sub   r13,r12
     rra   r12
     rra   r12
     mov   r12,r14
     rra   r12
     rra   r12
     rra   r12
     rra   r12
     sub   r12,r14         ;N1 in r14
     add   r14,r11         ;N3=T2+N1 in r11
     add   r14,r13         ;N2=T1+N1
     mov   r13,T2_770
           ;"Common" Lossy-Adaptor
           ;expects N3 in r11
           ;1/n1=32, gamma_2=-1+1/32)
           ;n1*gamma_1=1/32
     mov   r11,r13        ;save N3 to r13
     mov   IN,r12         ;T1=N3*gamma_2+n1*gamma_1*IN
     rra   r12
     rra   r12
     rra   r12
     rra   r12
```

```
     rra   r12           ;n1*gamma_1*IN in r12
     sub   r11,r12
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     rra   r11
     add   r11,r12       ;T1 in r12
     mov   r12,T1_770    ;save T1 to RAM

     rla   r13           ;N3 * 32
     rla   r13
     rla   r13
     rla   r13
     rla   r13

     rla   r12           ;T1 * 32
     rla   r12
     rla   r12
     rla   r12
     rla   r12
     sub   IN,r12

     add   r12,OUT       ;r13 is OUT
           ;*************************************
     cmp   OUT,MAXLO     ;compare Output with MAX
     jge   SWAP7
     mov   r13,MAXLO     ;big value to MAX
     mov   #04,ROW       ;set Row to 4
SWAP7      ;*************************************
           ;Common-Lossy End
FILTER_770_ENDE
           .endif

           .if FILTER_8
FILTER_697
     mov   T2_697,r11
     mov   r11,r12
     mov   T1_697,r13
     sub   r13,r12
     rra   r12
     rra   r12
     mov   r12,r14
     rra   r12
     add   r12,r14
     rra   r12
     rra   r12
     rra   r12
     sub   r12,r14       ;N1 in r14
     add   r14,r11       ;N3=T2+N1 in r11
     add   r14,r13
     mov   r13,T2_697
           ;"Common" Lossy-Adaptor
```

```
            ;expects N3 in r11
            ;1/n1=32, gamma_2=-1+1/32)
            ;n1*gamma_1=1/32
      mov   r11,r13       ;save N3 to r13
      mov   IN,r12        ;T1=N3*gamma_2+n1*gamma_1*IN
      rra   r12
      rra   r12
      rra   r12
      rra   r12
      rra   r12           ;n1*gamma_1*IN in r12
      sub   r11,r12
      rra   r11
      rra   r11
      rra   r11
      rra   r11
      rra   r11
      add   r11,r12       ;T1 in r12
      mov   r12,T1_697    ;save T1 to RAM
      rla   r13           ;N3 * 32
      rla   r13
      rla   r13
      rla   r13
      rla   r13
      rla   r12           ;T1 * 32
      rla   r12
      rla   r12
      rla   r12
      rla   r12
      sub   IN,r12
      add   r12,OUT       ;r13 is OUT
            ;***********************************
      cmp   OUT,MAXLO     ;compare Output with MAX
      jge   SWAP8
      mov   OUT,MAXLO     ;big value to MAX
      mov   #00,ROW       ;set Row to 0
SWAP8 ;***********************************
            ;Common-Lossy End
FILTER_697_ENDE
            .endif

FILTER_ENDE
      bis.b #LPM0,SR      ;CPU aus
      br    #DTMF_FILTER

;-----------------------------------------------------------------
;     8Bit-Timer Interrupt
;-----------------------------------------------------------------
TIM_8B
      bic   #PD,&ACTL     ;AD-Wandler an
      bis   #001h,&ACTL   ;starte AD-Wandlung
      reti
```

```
;***************************************************************
;      ADC Interrupt
;***************************************************************
AD_INT
     bis   #PD,&ACTL      ;AD-Wandler aus
     bic   #0f0h,0(SP)    ;AM to Stack
     reti

;***************************************************************
; LCD  Definitions
;***************************************************************
LCD_TYPE
a    .equ    01h
b    .equ    02h
c    .equ    10h
d    .equ    04h
e    .equ    80h
f    .equ    20h
g    .equ    08h
h    .equ    40h

;***************************************************************
;      LCD Table for DTMF Numbers
;***************************************************************
DTMF_Tab
        .byte   b+c           ; displays "1"
        .byte   a+b+d+e+g     ; displays "2"
        .byte   a+b+c+d+g     ; displays "3"
        .byte   a+b+c+e+f+g   ; displays "A"
        .byte   b+c+f+g       ; displays "4"
        .byte   a+c+d+f+g     ; displays "5"
        .byte   a+c+d+e+f+g   ; displays "6"
        .byte   c+d+e+f+g     ; displays "B" b
        .byte   a+b+c         ; displays "7"
        .byte   a+b+c+d+e+f+g ; displays "8"
        .byte   a+b+c+d+f+g   ; displays "9"
        .byte   a+d+e+f       ; displays "C"
        .byte   a+d+e+f+g     ; displays "E"
        .byte   a+b+c+d+e+f   ; displays "0"
        .byte   a+e+f+g       ; displays "F"
        .byte   b+c+d+e+g     ; displays "D" d

;***************************************************************
;      Interrupt Vector Table
;***************************************************************
       .sect "Int_Vect"       ,0ffe0h

       .word     INIT         ; Port0, bit 2 to bit 7
       .word     INIT         ; Basic Timer
       .word     INIT         ; no source
       .word     INIT         ; no source
       .word     INIT         ; Timer Port
       .word     AD_INT       ; EOC from ADC
```

```
        .word     INIT          ; no source
        .word     INIT          ; no source
        .word     INIT          ; no source
        .word     INIT          ; no source
        .word     INIT          ; Watchdog/Timer, Timer mode
        .word     INIT          ; no source
        .word     TIM_8B        ; 8b-Timer (P0.0 Int)
        .word     INIT          ; P0.0 Int.
        .word     INIT          ; NMI, Osc. fault
        .word     INIT          ; POR, ext. Reset, Watchdog
```

9 Hardware for coupling in signals

The conversion of the analog signals into digital values should be done with the ADC which is contained in the MSP430C25x. The reference voltage is applied to Pin SVCC. This can either be generated externally, or taken from the operating voltage of the controller (V_{DD}). The voltage range between 0 V and SVCC will then be resolved at up to 14 Bit. The operating voltage must be 5V, because of the high clock frequency of the CPU of 3.3 MHz. If this voltage is used as the reference voltage for the ADC and connected via an internal switch to Pin SVCC, then it is possible to do without an external reference voltage. One LSB then corresponds to a voltage of 0.305 mV. Commercially available DTMF receivers accept signal levels from about +1 dBm (600Ω) to -31 dBm (600Ω) This corresponds to amplitudes from 1.230 V down to 30.8 mV. The lowest voltage will thus still be resolved with about 100 steps. If still lower voltages need to be processed, an external voltage reference can be connected to Pin SVCC. A capacitor (C1) filters out interference from the external or internal reference voltage. To calculate the digital filter components, measurement values with signs are needed which the ADC can not supply. The analog input which is used is therefore biased by means of two close tolerance resistors (R1/R2) connected to SVCC/2, and the signal to be sampled connected in capacitively via C2. The numerical value of SVCC/2 is then subtracted from each digital input value. If the output value of the ADC in a quiescent state should give a numerical value of 01fffh with a resolution of 14 Bit, this being at the same time the voltage SVCC/2, then the remainder after subtraction of the numerical value of SVCC/2 is 01fffh-01fffh=0000h. Positive voltages thus result in positive numerical values, and *vice versa.* The coupling capacitor C2 and the resistors R3 + (R1 || R2) together form a high pass filter. If the cutoff frequency is fixed at about 200 Hz, lower frequency interference - for example mains (line) hum - will be suppressed. The ADC input is protected from excessive voltages by the resistor R3 and the Zener diode D1. The corresponding circuit diagram is shown in Figure 22:

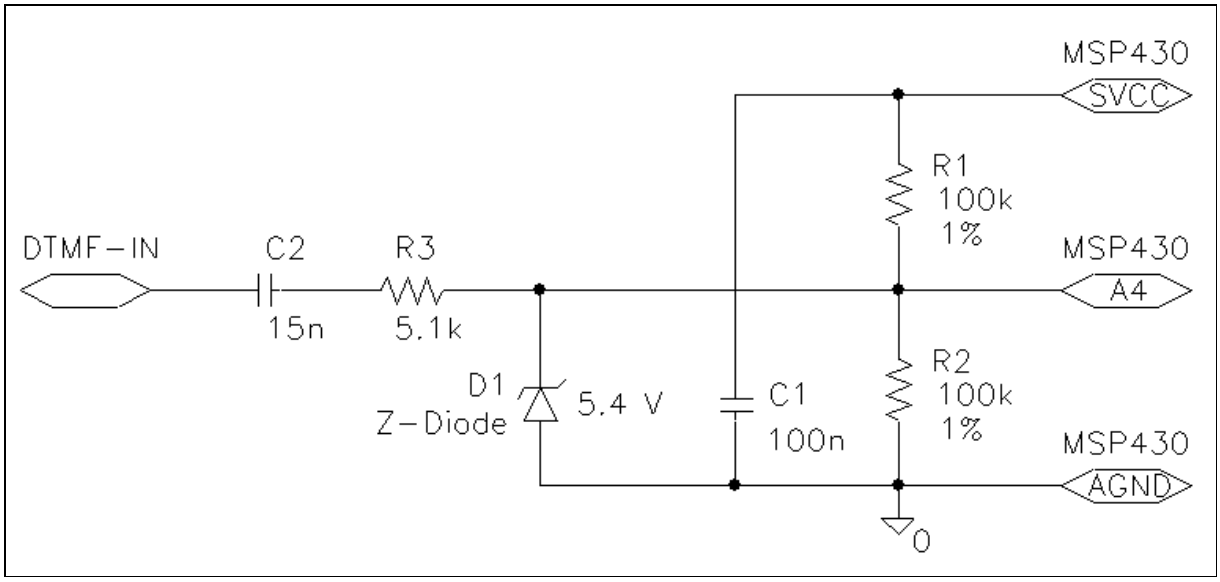

Figure 22: Circuit diagram showing coupling in of signals to the ADC

Since the measurement value with signs are only 14 Bit wide, whereas processing in the CPU can be performed with 16 Bit, before calculation the measurement values are shifted two places to the left, corresponding to a multiplication with 4. Due to this input scaling one takes better advantage of the available dynamic range and the noise caused by the rounding process during the multiplications is reduced.

10 Measurements and Results

Figure 23 shows the filtering duration for a sampling period. Channel 1 represents the ADC Interrupt; whenever an Interrupt occurs, a signal is delivered to a port pin. Channel 2 shows the duration in time of the filter calculation; a port is set at the beginning, and again cleared at the end of the algorithm.

The algorithm requires only 55% of the time which is available. The system frequency is 3.3 MHz. This results in a cycle time of 300 ns. There are therefore 915 Cycles available between two sampling periods. 474 Cycles are needed for the calculation of the 359 instructions. This results in an actual computing speed of 2.53 MIPS. As a result of the use of many registers/register instructions, an average of only 1.32 Cycles/Instruction are needed. Although the calculation speed is very high, the current consumpion of the application is only about 1.8 mA.

The software needs about 1 KB memory in the ROM and 32 Bytes in the RAM.

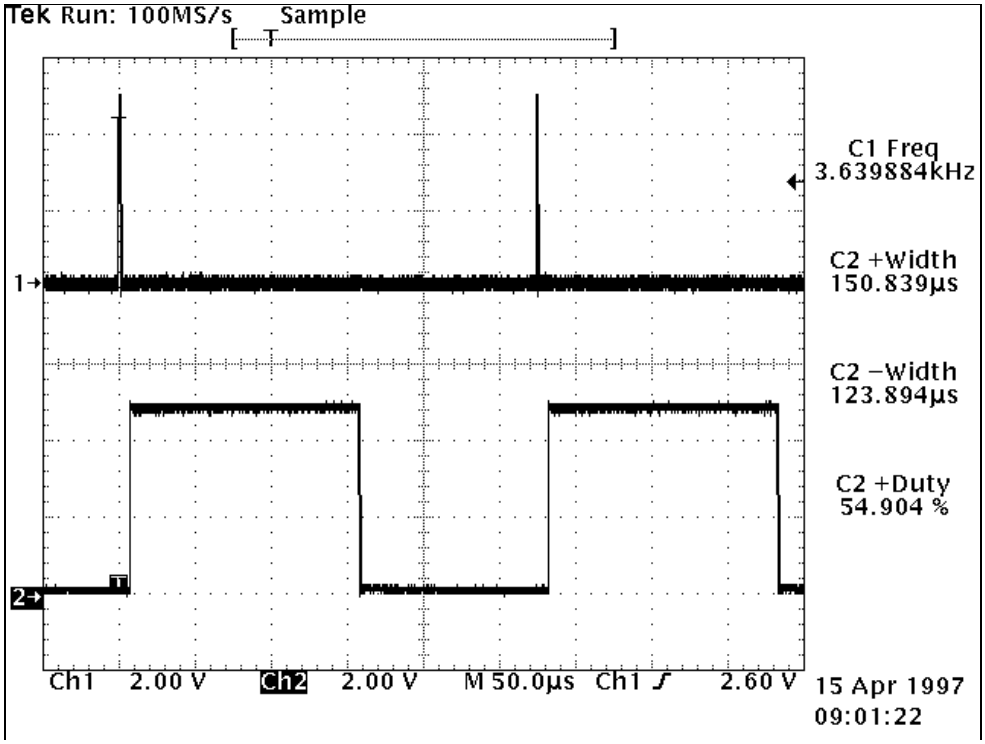

Figure 23: Duration of filter calculations resulting from sampling

11 Summary

The computation of real-time systems requires high computing speed, and DSPs are therefore often used for this purpose. The use of the MSP430 in this system for the calculation of digital filters is unusual, particularly since the multiplications must be performed without hardware multipliers. The MSP430 is however suitable for just those systems in which frequencies must be recognized from sampled values, and the calculated digital values no longer need to be taken via a D/A converter to the output. A high performance ADC has already been integrated into the MSP430. The resolution of 14 Bit guarantees high dynamic range for conversion and calculations.

In the application exemplified here, frequencies having input levels from 25 mV_{RMS} up to 1.5 V_{RMS} can be processed without problems. The resulting dynamic range is with more than 35 dB so great that commercially available DTMF receivers can easily be replaced. Since the 14 Bit ADC offers a theoretical dynamic range of 84 dB, there is no obstacle to extending the dynamic range for the recognition of signals further in both directions. This makes it possible, for example, to dispense with an internal reference voltage.

Depending on the complexity of the filter and the necessary sampling rate, signals of up to 5 kHz can be acquired with this system.

The development and implementation of eight narrow band bandpass filters which has been described here demonstrates:

- the suitability of the MSP430 and the built-in ADC for real-time filtering applications;
- the excellent properties of the wave digital filter;
- the efficiency of an average of 1.32 Cycles per instruction, made possible by RISC processor architecture and an orthogonal instruction set;
- and that 359 instructions are adequate for 8 DTMF filters.

12 References

[1] *Bundesamt für Post und Telekommunikation* (Federal Office for Post and Telecommunications): BAPT 223 ZV 5, *Zulassungsvorschrift für Endeinrichtungen zur Anschaltung an analoge Wählanschlüsse (ausgenommen Notruf- und Durchwahlanschlüsse) des Telefonnetzes* (approval specification for end equipment to be connected to analog dialing connections of the telephone network, except for emergency and in-dialing connections) / ISDN of the *Deutschen Bundespost Telekom; Bundesministerium für Post und Telekommunikation*, Draft, Bonn April 1994

[2] Tietze / Schenk: *Halbleiterschaltungstechnik*; (Semiconductor Circuit Design), 10th. Edition; Springer Verlag, Berlin 1993

[3] Marven / Ewers: A Simple Approach to Digital Signal Processing; Texas Instruments, 1994

[4] Sauvagerd, Ulrich: A Ten-Channel Equalizer for Digital Audio-Applications; IEEE Transactions on Circuit and Systems, Vol. CAS-36, No 2, February 1989, pp. 276-280

[5] Kaiser, Ulrich: RISP: *Eine digitale Signalprozesorarchitektur mit reduziertem Befehlssatz für Wellen-Digitalfilter* (Digital Processor Architecture with Reduced Instruction Set for Wave Digital Filters); Dissertation, *Fakultät für Elektrotechnik, Ruhr-Universität* Bochum 1991

[6] Lutz Bierl / Texas Instruments: MSP430 Family, Metering Application Report, Texas Instruments, Version 2.1, Jan 1996

[7] Texas Instruments: MSP430 Family, Architecture User's Guide and Module Library, Texas Instruments, 1996

[8] Texas Instruments: MSP430 Family, Software User's Guide, Texas Instruments, 1996

[9] Texas Instruments: MSP430 Family, Assembly Language Tools User's Guide, Texas Instruments, 1996

[10] Kitzberger, Holger: *Aufbau einer Signalprozessorkarte mit MSP430 und Untersuchung eines Wellen-Digitalfilters*; (Design of a signal processor card with the MSP430, and tests on a wave digital filter) *Diplomarbeit, Fachhochschule* Regensburg, April 1994

[11] Siwy, Robert: *Systementwicklung einer Telekom-Applikation zum Senden und Empfangen von DTMF-Signalen mit dem Microcontroller MSP430* (System development of a telecom application to send and receive digital signals with the Microcontroller MSP430); *Diplomarbeit, Fachhochschule* Landshut, Mai 1997

Additional Literature

All details refer to a list of additional literature provided by Ulrich Kaiser:

{5} Fettweis, A.: Wave Digital Filters: Theory and Practice; IEEE Proceedings, Vol. 74, No. 2, Feb. 1986, pp. 270-327

{107} Kaiser, Ulrich: *Wellen-Digitalfilter - Prädestiniert für Kundenspezifische digitale Signalverarbeitung* (Wave digital filters: predestined for customer specific digital signal processing); ELEKTRONIK 26/1988 , pp. 49-53 and 1/1989, pp. 50-54

{12} Gaszi, L.: DSP-Based Implementation of a Transmultiplexer using Wave Digital Filters; IEEE Trans. on Communications, Vol. COM-30, No. 7, pp. 1587-1597, July 1982

{22} Wegener, W.: *Entwurf von Wellen-Digitalfiltern mit minimalem Realisierungs-aufwand* (Design of wave digital filters with minimum effort); Dissertation, *Abteilung für Elektrotechnik, Ruhr-Universität* Bochum 1980

{120} Claesen, T. / Mecklenbräuker W. / Peek, J.: On the Stability of the Force Response of Digital Filters with Overflow Nonlinearities, IEEE Trans. Circuits and Systems, Vol. CAS-22, August 1975, pp. 692-696

{136} Cervera, A.: *Untersuchungen der Stabilität konventioneller rekursiver Digitalfilter unter Schleifenbedingungen* (Investigation of the stability of conventional recursive digital filters under loop conditions); Dissertation, *Ruhr-Universität* Bochum 1987

{19} Fettweis, A., Meerkötter, K.: On Parasitic Oscillations in Digital Filters under Looped Conditions, IEEE Trans. on Circuits and Systems, Vol. CAS-24, No. 9, September 1977, pp. 475-481

{228} Kaiser, U., Wave Digital Filtering for TI's Sensor Signal Processor MSP430, TI Technical Journal, Vol. 11, No. 6, Nov. 1994, pp. 65-83

{235} Kaiser, U., Wave Digital Filter Implementations on the Low-Power Sensor Signal Processor MSP430, Proc. of European Conference on Circuit Theory and Design, ECCTD'97, 1.-3.Sept.1997, Budapest, pp. 777-782