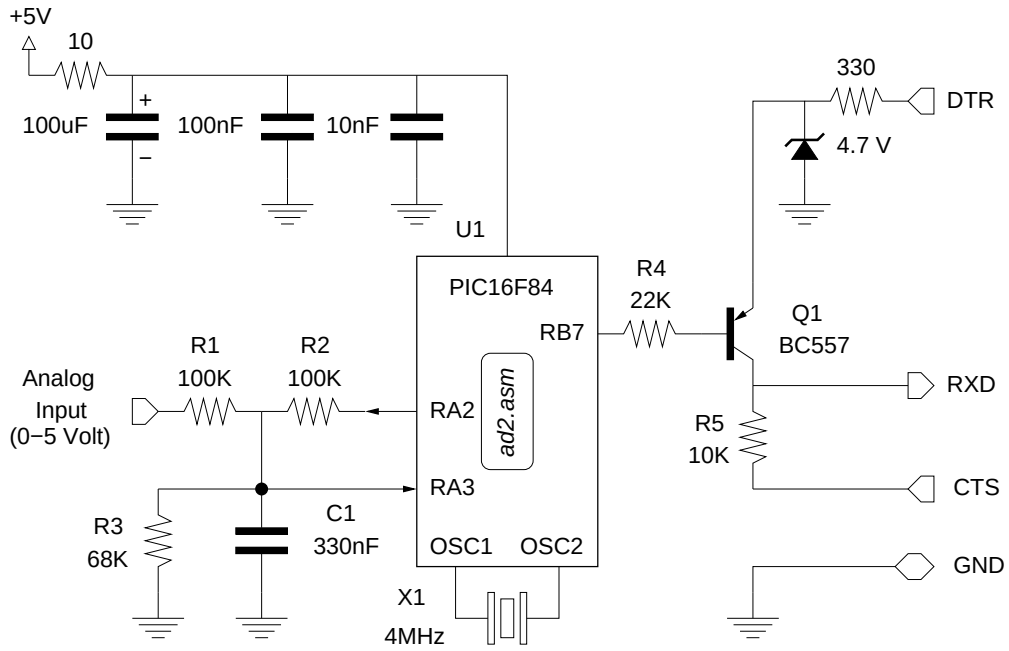
10-bit ΔΣ ADC from a PIC16F84

Jesús Arias

22nd November 2003

1 The circuit



The schematic of the ADC consist mainly of a PIC16F84 microcontroller and a RC network. The microcontroller generates an RS232-compatible output (9600 baud, 8 bit, no parity) thanks to the level-converter circuit that is shown on the right of the schematic. The PIC16F84 must be clocked at 4 MHz.

The ratio between R1 and R2 defines the input range of the converter:

$$\Delta V_{in} = \frac{R2}{R1} V_{dd}$$

The resistor R3 is used to adjust the input offset. The center-of-scale code is obtained for the following input voltage:

$$V_{in0} = V_{th}\left(1 + \frac{R1}{R2} + \frac{R1}{R3}\right) - \frac{V_{dd}}{2}\frac{R1}{R2}$$

were V_{th} is the threshold voltage of the input pin (RA3). In our case V_{dd} is 5 volts and V_{th} is about 1.4 volts (from PIC16F84 datasheet). Alternatively, if we want our input range centered around $V_{dd}/2$, the R3 value must be:

$$R3 = (R1 \parallel R2)\frac{V_{th}}{V_{dd}/2 - V_{th}}$$

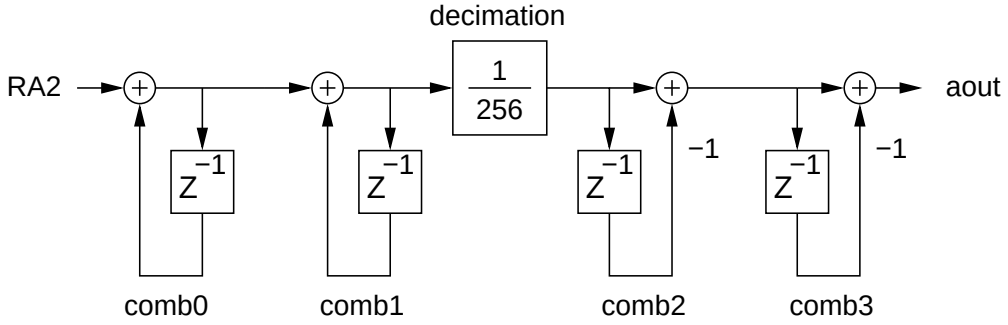
The resistor values in this design give an input range of 5 volts centered around 2.5 volts.

The $\Delta\Sigma$ conversion works generating a train of pulses on the pin RA2 whose average value cancels the input signal at the capacitor node. This forces a voltage at pin RA3 very close to the pin threshold. The program running in the PIC microcontroller generates the output pulses by just inverting the logic level of pin RA3 every 104 μs and copying it into the RA2 output. These pulses are recorded and they are used to reconstruct the equivalent input voltage by doing a digital filtering.

A good, regulated, power supply is required. Decoupling capacitors must be close to microcontroller power pins. In addition, we must avoid loading other microcontroller's pins. e.g. It would be a bad idea to drive a LED directly from a microcontroller pin. If other pins are used as outputs they must be isolated using a buffer powered from a different voltage source in order to maintain the microcontroller power supply clean.

2 The digital filter

The filtering of the $\Delta\Sigma$ modulator stream is done by program. The filter used is a second-order comb filter, whose structure is:

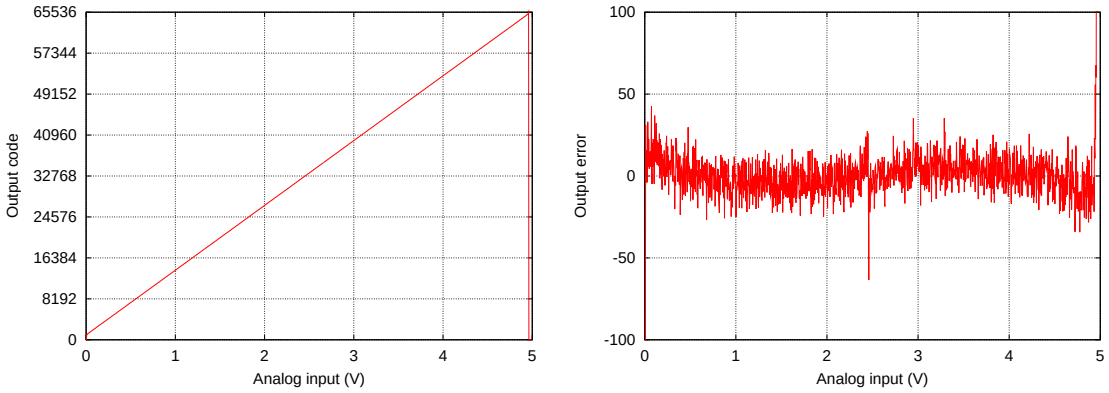


This filter is composed of two integrators, running at 9600 Hz, followed by sample decimation and two differentiators running at 37.56 Hz. The filter input is only one bit, but each "comb" variable is 16 bits long and so it is the output. It must be noted that, even if the output samples are 16-bit long, the less-significant bits are almost random. The expected AD resolution is only 10 bits. The data processing is done in an interrupt routine using 16-bit integer arithmetic (A pain in the neck for a CPU without ADD-with-carry instructions).

3 Simulation results

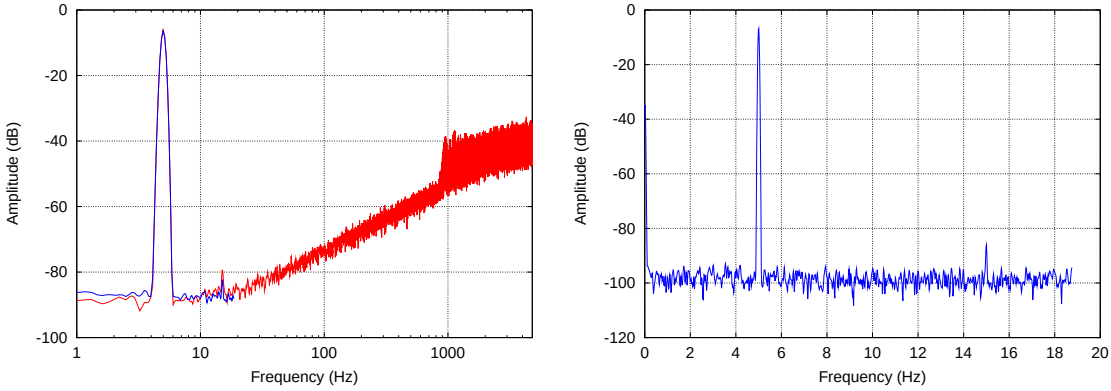
The following results were obtained using the "f84" simulator conveniently modified to simulate the analog section on the prototype and its serial output.

3.1 Transfer function & error



A linear voltage ramp is applied to the input. This ramp starts at 0 volts and it ends at 5 volts. The time interval is 40 sec long or 1502 samples.The simulated transfer function is an almost perfect straight line, but there is still a small offset error (The code for 0 volts is not zero). Another interesting property is that a positive overflow gives a zero code instead of 65535 (in fact it is a truncated 65536). The error graph on the right is the difference between the actual ADC output and the best straight-line fit. Here, a small nonlinearity is observed. The error amplitude is about 30 LSBs, or 2000 times smaller than the output range. This error is bigger for input voltages around 2.5 V due to a remaining tonal behavior (The $\Delta\Sigma$ output is a perfect square wave for that input).

3.2 Spectra & SNR



In this simulation the input signal is a sinusoid centered around 2.5 volts. The first graph shows the spectra of the pulse train at RA2 (red line) and the ADC output (blue line). The sampling frequency is 9615 Hz for the pulse output and 37.65 Hz (9615/256) for the ADC output. Here we can see that the digital filter does a good job removing the high-frequency noise and lowering the sample rate without allowing too much noise to fold back in top of the signal band. The FFT length was 65536 samples for RA2 output and 256 samples for ADC output.

The signal-to-noise ratio (SNR) was estimated at 65dB from this graph. This means that the effective ADC resolution is about 10.5 bits ($ENOB = (SNR - 1.76)/6.02$)

The second graph shows only the ADC output. In this case we want to see how high is the distortion of the input signal. We can see that the third harmonic of the input signal is attenuated more than 70 dB with respect to the input

tone. This means that the signal distortion is less than the ADC noise, and, therefore, we can state that this is a very linear converter. The FFT length was 2028 samples.

4 The program

The microcontroller program is written in native assembly language. It requires 166 words of flash memory for the program itself and 31 bytes of RAM for variables. Interrupts are called every 104 clock cycles. A typical interruption takes 39 cycles to execute, but every 256 interrupts we have a longer interruption that takes 67 clock cycles. The AD conversion accounts for a 37% of the total CPU time.

5 Program listing

```
;-------------------------------------------------
; This program is distributed under GNU GPL license
;-------------------------------------------------
        processor p16f84
        __config 0x3ff2
        radix   dec

w       equ     0
f       equ     1

c       equ     0
dc      equ     1
z       equ     2

indf    equ     0x0
tmr0    equ     0x1
pcl     equ     0x2
status  equ     0x3
fsr     equ     0x4
porta   equ     0x5
portb   equ     0x6
eedata  equ     0x8
eeadr   equ     0x9
pclatch equ     0xa
intcon  equ     0xb

optionr equ     0x81
trisa   equ     0x85
trisb   equ     0x86
eecon1  equ     0x88
eecon2  equ     0x89

;-----------------------------------------------
; Variable defs.
```

```
;-----------------------------------------------
        cblock  0xc

        tmp_w
        tmp_st
        nirq
        nsamples
        sampleix
        comb0
        comb0h
        comb1
        comb1h
        comb2
        comb2h
        comb3
        comb3h
        aout
        aouth
        adata
        adata_h
        nchar
        resto
        cntbit
        obyte
        uart
        uarth
        uartdel
        tmp
        buf

        endc

;------ RESET ---------------------------
;       Configuration:
;       ---------------
;       RA3 A/D input
;       RA2 A/D output
;       RB7 UART output
;
        org     0

        movlw   0xfb    ;RA2 out
        tris    porta
        goto    ini

;------ INTERRUPT (9615/sec)-----------------

        org     0x4
```

```
        movwf   tmp_w           ;save CPU status
        swapf   status,w
        movwf   tmp_st

        bcf     intcon,2        ;clear interrupt

        ;Interrupt latency equalization (to add 1-cycle delay if needed)
        ; - can give 3 dB improvement (1/2 bit)
        ;  Some experimentation is still needed because
        ;  PIC16F84 interrupt details are not well known
        ;We can try:
        ; - No equalization
        ; - btfsc   tmr0,0
        ; - btfss   tmr0,0
        btfsc   tmr0,0
        goto    deleq

deleq   movlw   259-104         ;adjust timer for a new
        addwf   tmr0,f          ;interrupt

        ;------ A/D -----
        rrf     porta,w         ;A/D feedback loop
        xorlw   4               ;~RA3 -> RA2
        movwf   porta

        ;---- UART ----         ;16-bit shift reg.
        bsf     status,c
        rrf     uarth,f
        rrf     uart,f
        btfsc   status,c
        bsf     portb,7
        btfss   status,c
        bcf     portb,7
        movf    uartdel,f         ;dec.  uartdel if not 0
        btfss   status,z
        decf    uartdel

        ;---- Comb Filter:  integrators ----
        btfsc   porta,2
        goto    isr1
        incfsz  comb0,f
        goto    isr1
        incf    comb0h,f
isr1    movf    comb0,w
        addwf   comb1,f
        btfsc   status,c
        incf    comb1h,f
```

```
movf    comb0h,w
addwf   comb1h,f

;----- DECIMATION: The differentiator code is called
;----- every 256 interrupts

incfsz  nirq,f
goto isrf

;----- Comb Filter:  differentiators -----
; aout =comb1-comb2-comb3
; comb3=comb1-comb2
; comb2=comb1

movf    comb1,w
movwf   aout
movf    comb1h,w
movwf   aouth

movf    comb2,w
subwf   aout,f
btfss   status,c
decf    aouth,f
movf    comb2h,w
subwf   aouth,f

movf    aout,w
movwf   comb2
movf    aouth,w
movwf   comb2h

movf    comb3,w
subwf   aout
btfss   status,c
decf    aouth,f
movf    comb3h,w
subwf   aouth,f

movf    comb2,w
movwf   comb3
movf    comb2h,w
movwf   comb3h
movf    comb1,w
movwf   comb2
movf    comb1h,w
movwf   comb2h

incf    nsamples,f
```

```
        ;---- End of Interrupt ----
isrf    swapf   tmp_st,w        ;restore cpu status
        movwf   status
        swapf   tmp_w,f
        swapf   tmp_w,w
        retfie

;-----------------------------------------
;------ MAIN -----------------------------
;-----------------------------------------

ini     movlw   0x7e    ;RB7, RB0 out
        tris    portb

        movlw   0x88    ;No Pull-UPs, no prescaler
        option

        clrf    nirq    ;Clearing variables
        clrf    nsamples
        clrf    sampleix
        clrf    comb0
        clrf    comb0h
        clrf    comb1
        clrf    comb1h
        clrf    comb2
        clrf    comb2h
        clrf    comb3
        clrf    comb3h
        movlw   0xff
        movwf   uart
        movwf   uarth

        bsf     portb,7 ; Default state of UART's line

        movlw   0xa0    ; Interrupt on timer overflow
        movwf   intcon

buc     call    getad   ; get AD sample
        call    prtn    ; And print it in decimal
        goto    buc

;-------- UART EMULATOR (9600 bps) ---------
; Sends the byte:  "obyte"
;-----------------------------------------

putbyte movf    nirq,w
        movwf   tmp
```

```
pbbuc1  movf    tmp,w   ;wait until next interrupt
        subwf   nirq,w
        btfsc   status,z
        goto    pbbuc1

pbbuc2  movf    uartdel,f
        btfss   status,z
        goto    pbbuc2

        movlw   0xff    ; Filling the UART shift reg.
        movwf   uarth
        movf    obyte,w
        movwf   uart
        bcf     status,c
        rlf     uart
        rlf     uarth

        movlw   10      ; Wait until 10 interrupts
        movwf   uartdel

        return

;------------------ BIN to ASCII -----------------
; divide by 10:  adata(16 bits)/10 -> adata,resto
;-------------------------------------------------

div10   movlw   16
        movwf   cntbit
        clrf    resto
        movlw   10
bdiv1   bcf     status,0
        rlf     adata,f
        rlf     adata_h,f
        rlf     resto,f
        subwf   resto,f
        btfsc   status,c
        incf    adata
        btfss   status,c
        addwf   resto,f
        decfsz  cntbit,f
        goto    bdiv1
        retlw   0

;-------------------------------------------------
;-- prtn:  prints adata (16 bits) in ascii
;-------------------------------------------------
```

```
prtn    movlw   buf
        movwf   fsr
        clrf    nchar
bprtn1  call    div10
        incf    nchar
        movf    resto,w
        addlw   48
        movwf   indf
        incf    fsr
        movf    adata,w
        iorwf   adata_h,w
        btfss   status,z
        goto    bprtn1
bprtn2  decf    fsr,f
        movf    indf,w
        movwf   obyte
        call    putbyte
        decfsz  nchar,f
        goto    bprtn2
        movlw   10
        movwf   obyte
        call    putbyte
        retlw   0

;-----------------------------------------------
; A/D read:  new sample -> adata
;-----------------------------------------------

getad   movf    nsamples,w
        subwf   sampleix,w
        btfsc   status,z
        goto    getad
        movf    nsamples,w
        movwf   sampleix
        movf    aout,w
        movwf   adata
        movf    aouth,w
        movwf   adata_h
        bsf     portb,0 ; Pulse on PB0 for timming debug
        bcf     portb,0
        return
```

6 Experimental results

The proposed ADC was built and tested. First, in order to test the basic operation of the converter, an exponential ramp was applied to the input from a big capacitor that was charged from ground to +5V through a resistor. The recorded transient is shown in Fig 1.

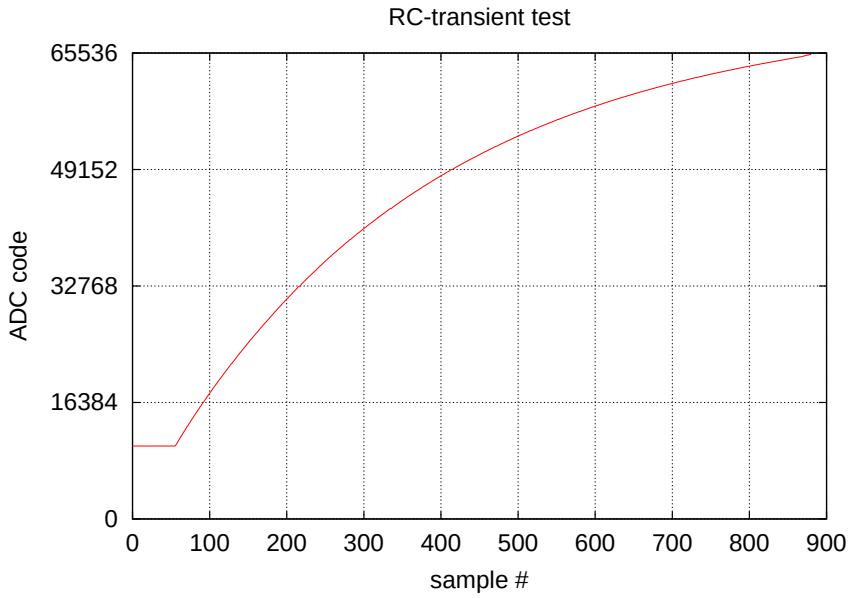


Figure 1: Transient signal recorded using the proposed ADC.

Then, by cropping the exponential part of the transient and by doing a nonlinear curve fitting we obtained the average error of the converter (Fig 2). This plot shows a different behavior than predicted from simulations, with an average error about 1 10-bit LSB and with higher peak values. The peaks in this plot are due to idle tones in the converter, a well known problem of first-order delta-sigma modulators.

Finally, a 4.7 Hz sinusoid was applied to the input and the recorded data stream was processed in the frequency domain using a Fourier transform. The obtained spectrum is shown in Fig 3. The measured SNR was 64 dB (10.3 effective bits) and the total harmonic distortion was -61 dB.

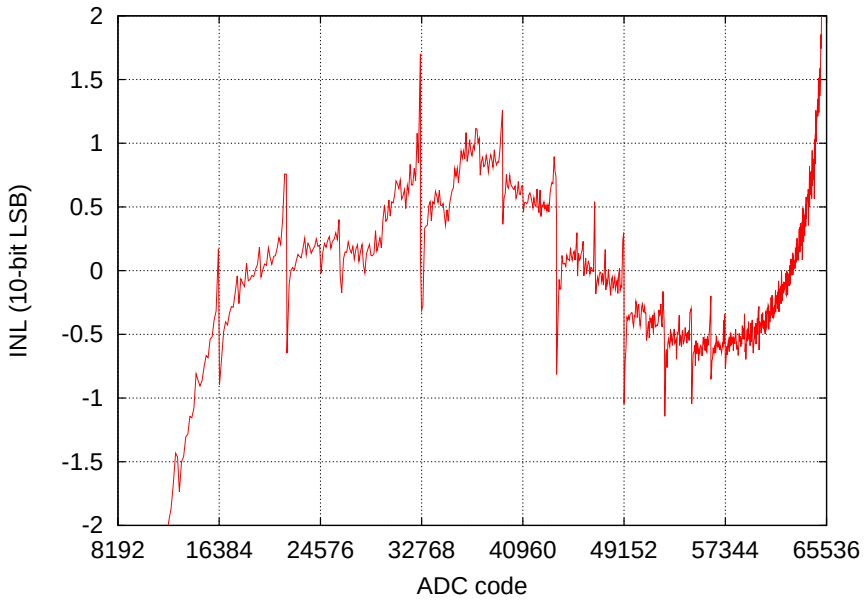


Figure 2: Nonlinearity plot of the ADC.

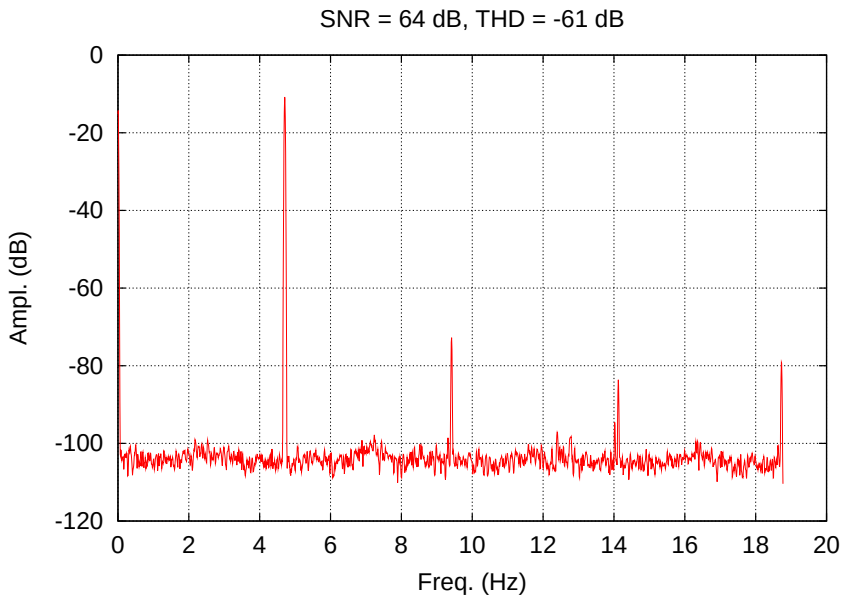


Figure 3: Spectrum of a digitized sinusoid. (4096-sample FFT)