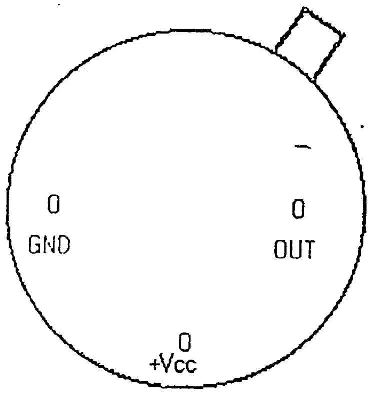
SMARTEC TEMPERATURE SENSOR -
SMT160-30

183113

MOST IMPORTANT FEATURES

* No A-D converter necessary
* Linear output within 0.2° C
* Output fully digital interpretable
* Output fully analogue interpretable
* Calibrated on chip
* TTL, CMOS compatible
* Short-circuit protected
* 5-Volt operation
* Temperature range 160° C (-30° to +130° C)
* No external parts required
* Low power consumption (<200 μA)
* Low noise



Bottom view
TO-18 Housing

PRODUCT HIGHLIGHTS

1. The SMART TEMPERATURE SENSOR represents a significant totally new development in transducer technology. Its new on-chip interface responds to the progressively stringent demands of both the consumer and industrial electronics sectors for a temperature sensor directly connectable to microprocessor input and thus capable of direct and reliable communicatoin with microprocessors.

2. The SMART TEMPERATURE SENSOR features a duty-cycle modulated square-wave output voltage with linear response to temperatures in the -30° C to +130° C range to an accuracy of better than .5° C. In the range from -30 to +100° C the linearity is better than 0.2° C.

3. The sensor is calibrated during testing and burn-in of the chip. The integral modulator ensures that the sensor unit can communicate effectively with low-cost processors without the need of expensive (on-board) A/D convertors.

4. The SMART SENSOR combines digital output and on-chip calibration to ensure major cost reductions and performance-related advantages.

5. Direct connection of the sensor output to the microprocessor input reduces the number of components and terminals to a minimum, cutting costs and boosting reliability.

6. Integral sensor calibration yields significant cost reductions and enhances performance.

7. Since the sensor requires no subsequent calibration, optimal cost savings are recorded both during manufacturing and in the course of after-sales servicing.

The SMARTEC SMT160-30 is available in TO-18-packaging. Chip size: 1.55 x 2.50 mm.

PRODUCT DESCRIPTION

The SMT160-30 is a three terminal integrated temperature sensor, with a duty-cycle output. Two terminals are used for the power supply of 5 Volts and the third terminal carries the output signal. A duty cycle modulated output is used because this output is interpretable by a microprocessor without A-D converter, while the analogue information is still available.

The SMT160-30 has an overall accuracy of 0.5° C in the range from -30° C to +100° C and a accuracy of 1.0° C from +100 to +130° C. This makes the sensor especially useful in all applications where "human" (climate control, food processing etc.) conditions are to be controlled.

Because of its direct connectability with microprocessors, the major part of an intelligent control loop is created by using only two components.

There where "intelligence" is already available the SMT160-30 can be used to safeguard temperature conditions. (Protection of overheating power transistors, motors and alike). Other applications are where absolute temperature influences parameters of certain components in a known way (eg. frequency of crystals, voltage regulators, precision amplifiers, mechanical components etc). In these applications the temperature drift can be compensated for by the microprocessor.

The C-mos output of the sensor can handle cable length up to 20 meters. This makes the SMT160-30 very useful in remote sensing and control applications.

TYPICAL APPLICATIONS

* Heater systems
* Air conditioners
* Climatizing units
* Washing machines
* Overheating protection
* Appliances

IMPORTANT PARAMETERS

Supply voltage	4.75 - 7 V
Supply current	max. 200 µA
Output Voltage	Vcc - 0.7 V
Short circuit protection	infinite (within supply voltage range)
Short circuit supply current	< 40 mA
Operating temperature range	-30 to +130° C
Storage temperature	-40 to +150° C

PERFORMANCE CHARACTERISTICS

Characteristic	min.	typical	max.	units
Supply voltage[1]	4.75	5	7	V.
Supply current			<200	µA
Temperature range[2]	-30	-	130	°C
Accuracy -30 +100° C				0.5°C
-40 +130° C				1.0°C
Calibration error	@23°C			0.25°C
Non-linearity[3]			0.2	°C
Supply voltage sensitivity				<0.1°C/V
Repeatability			0.2	°C
Long term drift		0.1		°C
Output:				
-frequency	1	-	4	Khz
-duty cycle				
Nominal @ 0°C		0.320		
Nominal temp coeff.		0.00470		/°C

[1] Case connected to ground

[2] The SMT 30-160-18 can be used from -65 to +160 °C for a short period without physical damage to the device. The specified accuracy applies only to the rated performance temperature range.

[3] Applicable from -30 to +100 °C

UNDERSTANDING THE SPECIFICATIONS.

The way in which the SMT160-30 is specified makes it easy to apply in a wide variety of different applications. It is important to understand the meaning of the various specifications and their effects on accuracy.
The SMT160-30 is basically a bipolar temperature sensor, with accurate electronics to convert the sensor signal into a duty cycle. During production the devices are calibrated including the converter.

THE OUTPUT SIGNAL.

As stated in the specifications the output is a square wave with a well-defined temperature-dependent duty cycle. The duty cycle of the output signal is linearly related to the temperature according to the equation:

D.C. = 0.320+0.00470*t   D.C. = duty cycle
t = Temperature in °C

Easy calculation shows that for instance that at 0 °C the D.C.= 0.320 or 32.0 % and at 130 °C the D.C.= 0.931 or 93.1 %

TOTAL ACCURACY.

The above mentioned equation is the nominal one. The maximum deviation from the nominal equation is defined as total accuracy. With temperatures above 100 °C the accuracy decreases.

NON LINEARITY

Non-linearity as it applies to the SMT160-30 is the deviation from the best-fit straight line over the whole temperature range. For the temperature range of -30 °C to +100 °C the non-linearity is less than 0.2 °C (TO18).

LONG-TERM DRIFT.

This drift strongly depends on the operating condition. At room temperature the drift is very low (< 0.05 °C is to be expected). However at higher temperatures the drift will be worse, mainly because of changes in mechanical stresses. This drift is partly irreversible and causes non-ideal repeatability and long-term effects. At temperatures above 100 °C but in the operating range a long-term drift better than 0.1 °C is to be expected.

NON LINEARITY

Non-linearity as it applies to the SMT160-30 is the deviation from the best-fit straight line over the whole temperature range. For the temperature range of -30° C to +100° C the non-linearity is less than 0.2° C.

LONG-TERM DRIFT

This drift strongly depends on the operating condition. At room temperature the drift is very low (<0.05° C is to be expected). However at higher temperatures the drift will be worse, mainly because of changes in mechanical stresses. This drift is partly irreversible and causes non-ideal repeatability and long-term effects. At temperatures above 100° C but in the operating range a long-term drift better than 0.2° C is to be expected.

GENERAL OPERATION

An easy way of measuring a duty cycle is to use a microcontroller. It is only necessary to connect the sensor´s output to one of the microcontrollers inputs. With help of a small program it is possible to sense that input wether it is high or low. The speed of this sampling is limited due to the instruction time of the controller. So to achieve the wished accuracy it is necessary to sample over more than one sensor period. This wax of working has also the advantage to filter noise. From the theory of signal processing it can be derived that there is a fixed ratio between the sensors signal frequency, the sampling rate and the the sampling noise. This sampling noise limits the accuracy and amounts to:

$$T_{error} = T_{range} * t_s/\mathrm{sqrt}(6*t_m*t_p)$$

T_{error}	=	measurement error (=standard deviation of the sampling noise)
T_{range}	=	considered temperature range
t_s	=	mirocontrollers sampling rate
t_m	=	total measurement time
t_p	=	output signal periodicity of the sensor

Modern microcontrollers can sample at a high frequency so with a small program it is possible to measure the sensor´s duty cycle within 50 ms and an accuracy of 0.01° C.

1. Introduction

The SMART temperature sensor transforms the ambient temperature to a square-wave TTL outputsignal with a temperature-dependent duty-cycle (see figure 1). Thanks to on-chip transformation, the sensor is able to communicate with low-cost microcontroller chips that do not contain an onboard A/D convertor. This note presents the theory of operation and some typical applications, including sample programs.

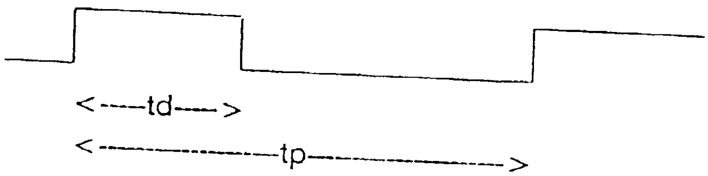


Fig. 1 $T = K1 * (td / tp) + K2$

2. Theory of operation

As stated above, the output-signals dutycycle is in linear relation to the ambient-temperature (equation 1). In order to measure the signals dutycycle, the signal must be sampled. The temperature can be derived by evaluating the measured sample-counts (equation 2). Temperature uncertainty can be calculated via equation 3. This equation shows that increasing the number of samples and/or decreasing the sample time will increase accuracy. The measurement time can be calculated from equation 4.

Equation nr 1 : $T = K1*(td/tp) + K2$

Equation nr 2 : $T = K1*(Nd/N) + K2$

Equation nr 3 : $dT = Trange/sqr(6*N*tp/ts)$

Equation nr 4 : $Tm \doteq N*ts$

3. Software programming

The following short pseudo-code program shows how to acquire the sample-counts and how to calculate the ambient-temperature. It should be noted that the software paths for both the 'high' and 'low' state of the output signal, are of equal code lenght. This is essential to secure an uniform sampling rate.

```
for i=1 to totalnumberofsamples
    begin
        if (SMARTsignal = high) then
            highcounter = highcounter + 1
        else
            lowcounter = lowcounter + 1
    end
AmbientTemperature = K1 * (highcounter/(totalNrOfSamples) + K2
```

Fig. 2 Pseudocode Smartec program.

4. Applications

A number of typical applications will be described in this section to illustrate how easy the sensor interfaces with all kinds of common microprocessors and how little software coding is needed to measure the ambient temperature. The description of examples includes calculation of measurement time, temperature accuracy, relevant hardware schematics and example programs in PLM and assembler.

4.1. MCS(r) - 96 Microcontrollers

The first example features an embedded controller application using the Intel MCS 8096 microcontroller chip. In this example, the SMART sensor output is connected to pin 1 of port 1.0 of the MCS 8096. The relevant hardware connection is shown in figure 4. It should be noted that the MCS 8096 has three eight-bit i/o ports; as a result, up to 24 sensors can be connected to one microcontroller.

The 8096 12 MHz microcontroller samples the sensor signal with a measured sample-rate of 8 usec. Substituting this rate in equation 3 results in a temperature accuracy of 0.1 C for approx. 13000 samples. The total measurement time is approx. 0.1 sec (equ 4).

The essential parts of the PLM-96 software program are listed below.

```
MEASURE_TEMPERATURE :
DO;
    declare     Temperature          word;
    declare     NrOfSamples          dword;
    declare     .......
    ........... etc

GetTemperature : procedure(TempPntr,TotalNrSamples);
    declare     TempPntr             word ;
    declare     TotNrSamples         dword;
    declare     (TotalCntr,LowCntr, HighCntr)     dword;

    TotalCntr = 0;
    HighCntr = 0;
    LowCntr = 0;

    DO TotalCntr = 1 to TotalNrSamples;
        IF      IOPORT0 AND 01H = 01H THEN
                HighCntr = HighCntr + 1;
        ELSE
                LowCntr = LowCntr + 1;
    CALL CalculateTemperature(TempPntr,highcntr,TotNrSamples);
end;
```

Fig. 3 Plm96 Listing.

The figure below shows the hardware schematics for a Smartec sensor connected to a 8096 microcontroller. Note that the connection illustrated is only one of many possible permutations.

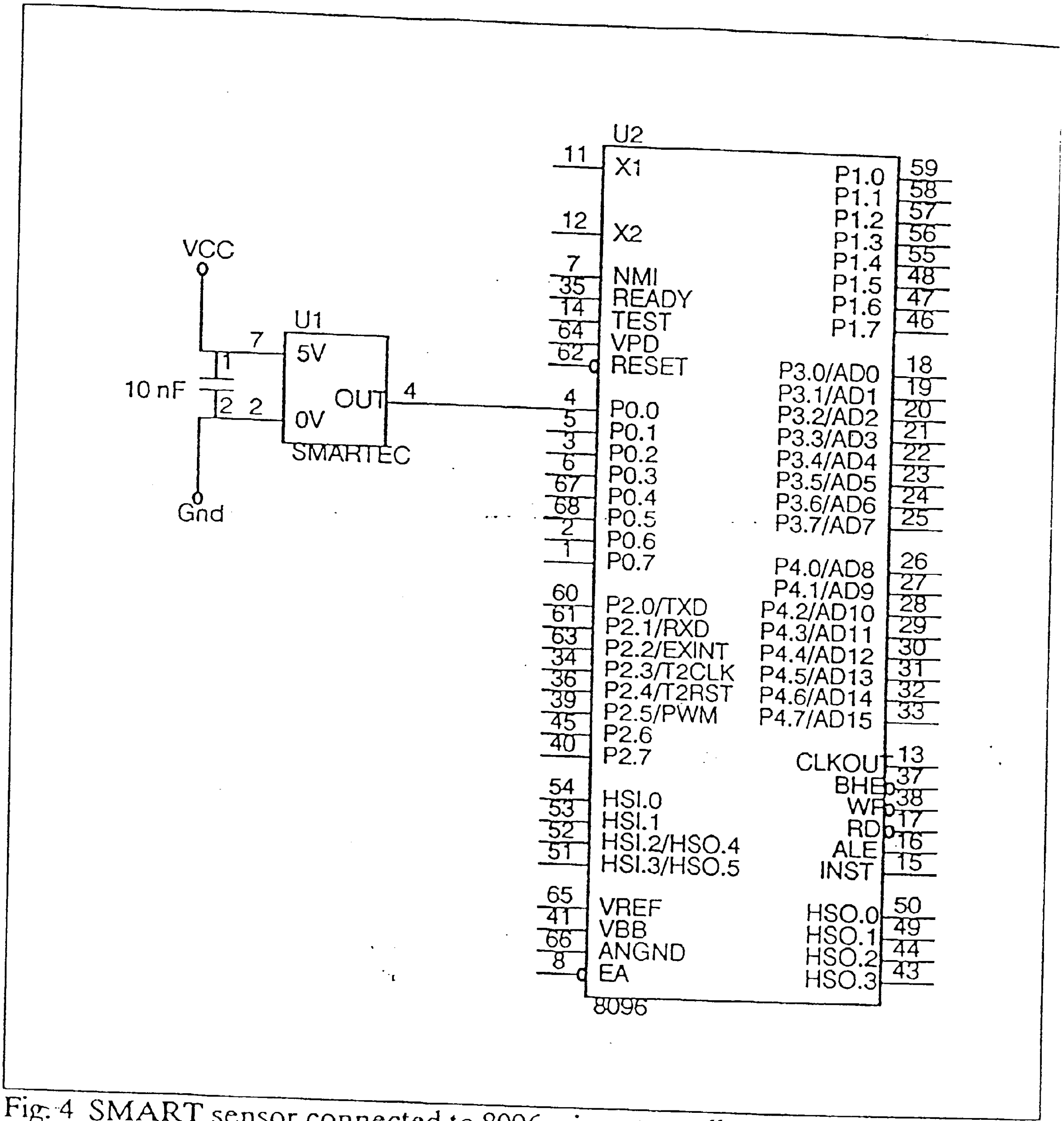


Fig. 4 SMART sensor connected to 8096 microcontroller.

4.2. MCS (r) - 51 Microcontrollers

The second example discusses an embedded controller application using the Intel MCS 8051 microcontroller chip. In this example, the SMART sensor output is connected to pin 1 of port 1.0 of the MCS 8051. The relevant hardware connection is shown in figure 5. It should be noted that the MCS 8051 has three eight-bit i/o ports; as a result 24 sensors can be connected to one microcontroller. The 8051 11 MHz microcontroller samples the sensor signal with a measured sample-rate of approx. 21 usec. Substituting this rate in equation 3 results in a temperature accuracy of 0.1 C for approx. 33000 samples. The total measurement time is approx. 0.7 sec (equ 4). The essential parts of the assembler software program are listed in figure 6.

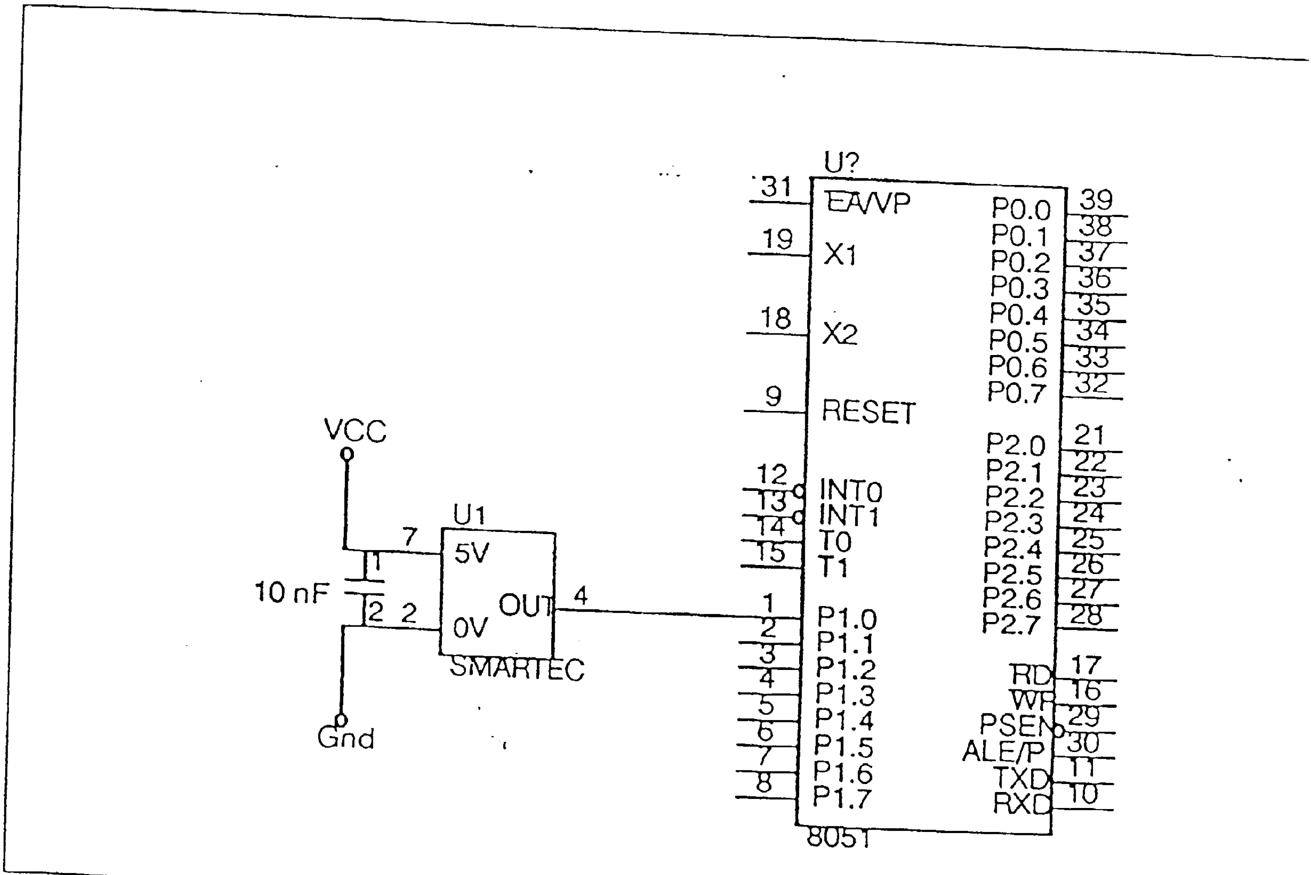


Fig. 5 SMART SENSOR connected to 8051 microcontroller

```
totalCount      EQU     50000D
BitMask         EQU     00000001B ;
IoPort          EQU     P1

START:
        MOV     R0,#00H
        MOV     R1,#00H
        MOV     R2,#00H
        MOV     R3,#00H
        MOV     R4,#00H
        MOV     R5,#00H

LOOPNOP:
        NOP                             ; DELAY 1 CYCLE
        NOP                             ; DELAY 1 CYCLE
LOOP:   MOV     A,(IOPORT)              ; Read port 0
        ANL     A,#BITMASK              ; Mask smartec bit
        JZ      INCLOW                  ; Jump if smartec signal is 0
INCHIG:
        INC     R2;                     ; Incr HighCount counter (low byte)
        CJNE    R2,#0,LOOP1NOP
        INC     R3                      ; Incr HighCount counter (high byte)
        JMP     LOOP1;
INCLOW:
        INC     R4                      ; Incr LowCount counter (low byte)
        CJNE    R4,#0,LOOP1NOP
        INC     R5; 1 CYCLE             ; Incr LowCount counter (High byte)
        JMP     LOOP1; 2 CYCLE

LOOP1NOP:
        NOP                             ; DELAY 1 CYCLE
        NOP                             ; DELAY 1 CYCLE
        NOP                             ; DELAY 1 CYCLE

;Increment totalcount and loop if not all done
LOOP1:  INC     R0                      ;
        CJNE    R0,#0,CHECKNOP          ;
        INC     R1                      ;
        JMP     CHECK                   ;

CHECKNOP:
        NOP                             ; DELAY 1 CYCLE
        NOP                             ; DELAY 1 CYCLE
        NOP                             ; DELAY 1 CYCLE

CHECK:  CJNE    R1,#HIGH(TotalCount),LOOPNOP
        CJNE    R0,#LOW(TotalCount),LOOP
READY:
        ; HighCount and Lowcount are known so the temperature can be calculated.
END
```

R0	: Low byte of TotalCount
R1	: High byte of TotalCount
R2	: Low byte of HighCount
R3	: High byte of HighCount
R4	: Low byte of LowCount
R5	: High byte of LowCount

Fig 6 ASM51 Listing

4.3. IBM PC/AT and compatibles.

The third example discusses an example where the SMART sensor output is connected to either the printer- or the game-port of an IBM PC. The require hardware connections for both situations are shown in figures 8 and 9. An advantage of the game-port over the printer-port is the availability of the +5 V powersupply pin. Supply voltage for the SMART sensor can't be generatec from the databits from the IBM printerport, because they don't meet the power-requirements of the sensor.
Up to 5 SMART sensors can be connected to 1 IBM printer port.
Up to 4 SMART sensors can be connected to 1 IBM game port.
The table 1 shows pins and pin names.

PRINTER PORT LPT1 LPT2 LPT3 or LPT4 (The base addresses of these ports can be found in the bios segment 0040H:0008 etc.)

Pinnr	11	10	12	13	15
Signalname	Busy	Ack	PE	Select	Error
Statusbyte	bit7	bit6	bit5	bit4	bit3

GAME PORT address 201 Hex

Pinnr	14	10	7	2
Signalname	Button D	Button C	Button B	Button A

Table 1

The formules as mentioned earlier can be used to determine the number of samples needed to get the desired accuracy. Essential parts of the Turbo Pascal software program are listed in figure 7..

```
Const     PortAdr    =     $201 ; {GamePort}
          Mask       =     $80  ; {bit 7 = button D}
Function GetTemperature:Real;
var
     value          :     byte     ;
     cnthigh        :     longint  ;
     totalcnt       :     longint  ;
     Dc             :     real     ;
     SampleCnt      :     longint  ;
Begin
     cnthigh     :=   0;
     For totalcnt     :=    1 to SampleCnt do
          cnthigh     :=    cnthigh + (port[PortAdr] and mask) ;
     cnthigh     :=   cnthigh div Mask ;
     Dc    :=    (cnthigh)/SampleCnt;
     GetTemperature:=     ((C2*DC) - C1)/100.;
End;
```

Fig 7 Turbo Pascal listing.

The following logic diagrams (fig.8 and 9) show how to connect a smartec sensor to the printerport and the gameport of an IBM pc. Please note that just 2 of many possible connections is shown in the logic diagrams.

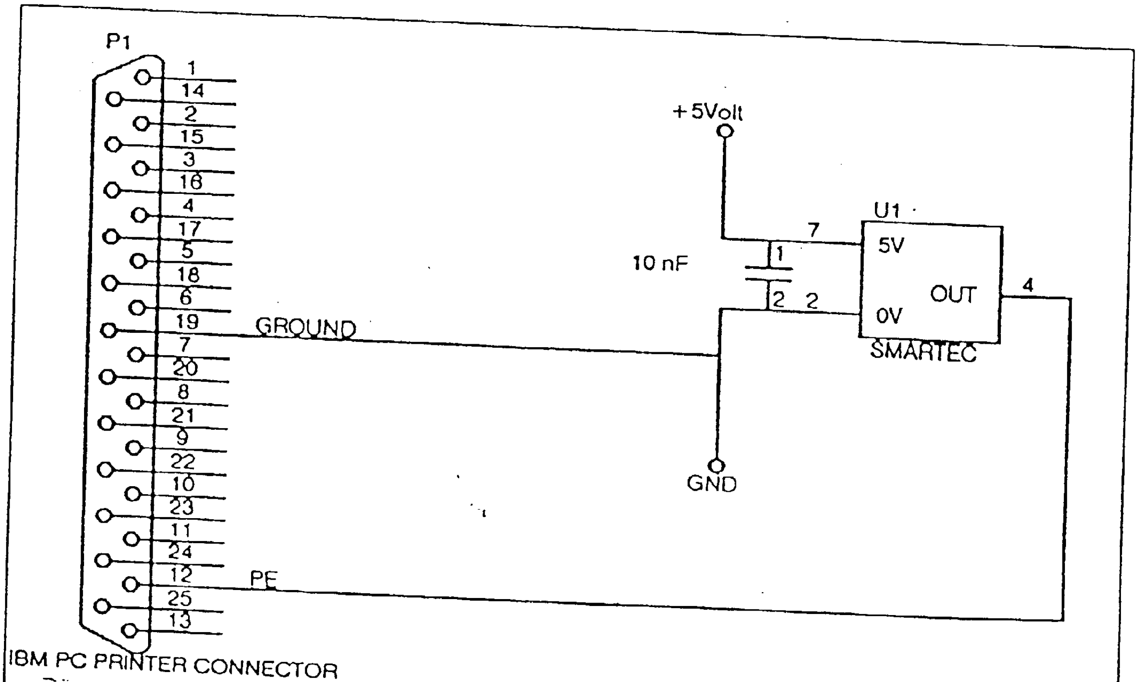


Fig.8 Smartec connected to IBM printerport.

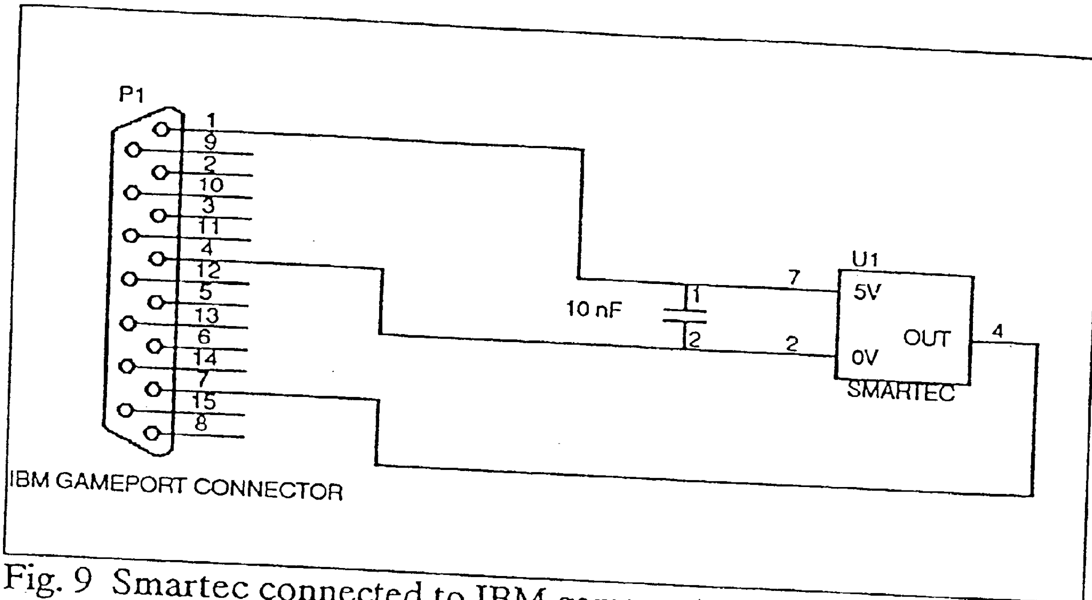


Fig. 9 Smartec connected to IBM gameport.

NOTE: It should be noted that a more accurate approach is to use a segment of assembler code in combination with your Pascal program. The reason is that the IF THEN ELSE construction is not symmetrical in execution time. The given assembler program is in fig. 10. The NOP's in the program are used to compensate for the difference in execution time of ,e.g., conditional jumps.

```
;
        .MODEL    TPASCAL
        LOCALS    @@
        .DATA
; *** THE EXTERNALS ARE DEFINED IN THE PASCAL PROGRAM ***
        EXTRN     TotalCnt : WORD
        EXTRN     CntLow   : WORD
        EXTRN     CntHigh  : WORD
        EXTRN     Masker     : BYTE
        EXTRN     PortAdr    : WORD
        .CODE
        PUBLIC    MyProc

ReadPortProc  PROC      NEAR
        push ax
        push cx
        push bx
        push dx
        push si
        mov       si,PortAdr
        mov       cx,TotalCnt      ; loopcount in cx
        mov       bx,0             ; CntLow  = 0 ;
        mov       dx,0             ; CntHigh = 0 ;
        CLI
@@LOOP:
        push dx
        mov       dx,si
        in        al,dx
        pop       dx               ;get PortAdr in al
        and       al,masker
        jnz       @@INCHIGH        ; 4 or 16 cycles
        nop                        ; 3 cycles
        nop                        ; 3 cycles
        nop                        ; 3 cycles
        nop                        ; 3 cycles
        inc       bx
        jmp       @@check
@@INCHIGH:
        inc       dx
        jmp       @@check
@@CHECK:
        loop      @@LOOP
        STI
        mov       CntLow,bx
        mov       CntHigh,dx
        pop       si
        pop       dx
        pop       bx
        pop       cx
        pop       ax
        ret
ReadPortProc        ENDP

        END
```

Fig. 10 IBM PC assembler listing.

MEASURING DUTY-CYCLES WITH AN INTEL 87C51 MICRO-CONTROLLER

The fastest way of measuring is with the aid of hardware. The 87C51 offers possibilities for that since it has 2 internal timer/counters.
2 port-pins (P3.2 and P3.3) can control these timer/counters directly by hardware. These I call "fast-inputs". All other pins can control the timer/-counters only by software.

There are 2 assembly programs written.
One measures the duty-cycle by hardware by using one of the fast inputs; the other program measures a duty-cycle by software.

A. HARDWARE CONTROLLED MEASUREMENT VIA P3.2

When the duty-cycle is measured by hardware there are some restrictions concerning the tasks the CPU is performing.
Both TIMER0 and TIMER1 are used. Normally TIMER1 is generating the baud rate for communication purposes. While measuring, the CPU is not allowed to transmit or receive any data.
Another restriction is that the fastest way of measuring is obtained by using interrupts which are preceded by the IDLE-mode of the CPU. During IDLE-mode, the CPU is non-active. The 2 timers are insensible to the IDLE command which is an important feature of the 87C51. The CPU will start up again by an interrupt. On this way, the CPU responds maximally fast on an interrupt. This special way of measuring demands the CPU not to run any background programs because that will cause errors in both the measuring and the background program.

Both timer/counters are selected in the 16-bits timer mode. Therefore the "timer/counter mode control" (TMOD-) register is initialized with the value 19H. In this mode TIMER0 only runs while P3.2 is logically "1", while TIMER1 can only be controlled by software. TIMER1 measures the measurement time. After a measurement the duty-cycle is obtained by:

$$\text{duty-cycle} = \frac{\text{contents\_of\_TIMER0}}{\text{contents\_of\_TIMER1}}$$

Detection of an edge with a resolution of 1µs is obtained when the measurement is started and stopped by using interrupts. An interrupt is generated on a falling edge of the input signal (when the interrupt flags are enabled).

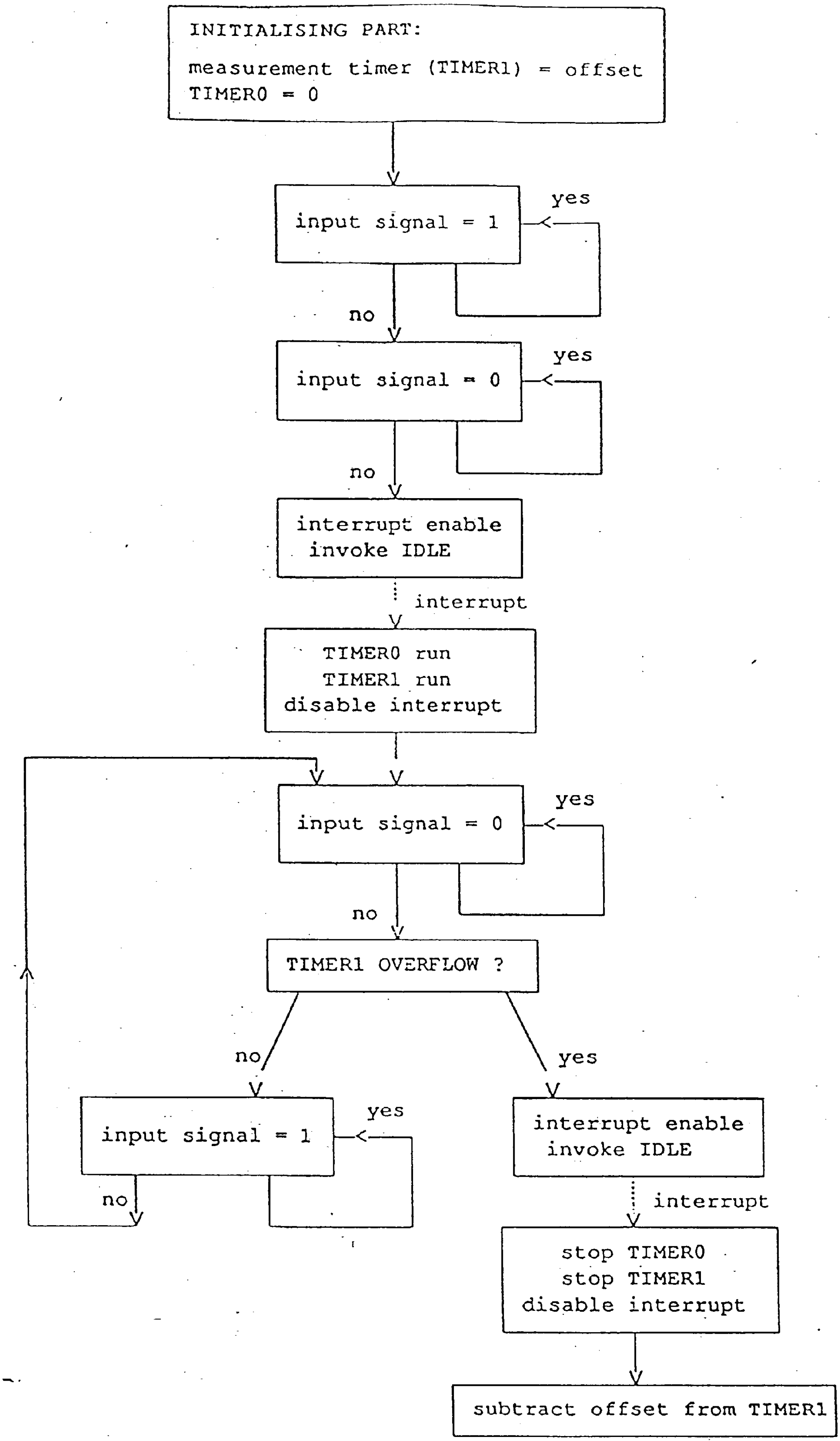


Figure 1 *Flow diagram of a duty-cycle measurement with the resolution of one machine cycle.*

The measurement is explained with the aid of a flow diagram (fig. 1).
First the contents of both timers are cleared. The initializing part starts with the detection of a 0-1 transition. Then the interrupt enable flag is set and the IDLE mode is invoked. Now the processor is waiting for an interrupt. When it is generated, the TIMER0 and TIMER1 run bits in the "timer control" (TCON) register are set. Now TIMER0 only runs when P3.2 is pulled high, so it measures the time that the input signal is logically 1. TIMER1 is running during the whole measurement. Besides the interrupt enable flag is cleared. The measurement is started.
Note that TIMER1 starts 3µs too late because processing of an interrupt takes 3µs. Because the measurement will be stopped at the same way this delay is eliminated.

Because of sampling noise it is known how many cycles of the input signal have to be involved within one measurement. In a software routine the number of periods can be kept in a PERIOD_COUNTER. This software routine is not affecting the sampling rate. Every period, PERIOD_COUNTER is decremented until it is zero. Then the measurement has to be finished. This PERIOD_COUNTER originates an inefficient use of the measurement timer (TIMER1). The period of the incoming temperature signal is between 300µs and 800µs. When a fixed number of periods is measured, TIMER1 doesn't always use all 16 bits. Then potential accuracy is spoiled. Therefore the measurement will be finished after TIMER1 generates an overflow. Now we have a 17 bits result which causes complex software routines. When TIMER1 is initialized with an offset, this offset will be subtracted from the 17 bits result so it will fit in 16 bits again. The measurement is now independent of the period.

When TIMER1 generates an overflow the measurement must be stopped. This will be done on the next 1-0 transition. First a 0-1 transition must be detected. Then the interrupt enable flag must be set and the IDLE mode will be invoked. After the interrupt both TIMER0 and TIMER1 run bits are cleared.

Now the contents of both timers (after correcting the offset of TIMER1) can be processed to the duty-cycle.

B. SOFTWARE CONTROLLED MEASUREMENT

Because it is impossible to use interrupts on all I/O ports except P3.2 and P3.3, it has to be possible to measure the duty-cycle by software as well. Again 2 counters are necessary. It is however still possible to use a hardware timer although it is software controlled. This timer can count the measurement time. An optimally fast software routine then measures the time when the input signal is logically "1". This is done by HIGH_COUNTER.

TIMER0 increments every machine cycle which is 1µs. The software sample rate is minimally 3µs. To obtain the duty-cycle, HIGH_COUNTER must be multiplied by 3:

```
;************************************************************************************
;                               TELLER IN R3_R4
;                               NOEMER IN R1_R2
SLOW:
;                               GEBRUIKT R5 LOKAAL VOOR OFSLAG
;************************************************************************************
RSEG ROM
        MOV TCON,#0
        MOV TLO,#0                      ;INITIALISE TIMERO
        MOV THO,#08H
        MOV TMOD,#01H                   ;OFFSET IN MEASUREMENT_TIME(RO)
        MOV R1,#0
        MOV R2,#0                       ;TEMPORARY HIGH COUNTER
        MOV R3,#0                       ;HIGH_COUNTER LOW BYTE
S1:     JNB P2.0,S1                     ;HIGH_COUNTER HIGA_BYTE
S2:     JB P2.0,S2
        SETB TCON.4
S3:     MOV A,R2                        ;TIMERO RUN
        ADD A,R1                        ;ADD HIGH_COUNTER AND TEMPORARY HIGH_COUNTER
        MOV R2,A
        MOV A,R3
        ADDC A,#0
        MOV R3,A
        MOV R1,#0
        JNB TCON.5,S6                   ;CLR TEMPORARY HIGH COUNTER
S4:     NOP                             ;TEST ON TIMER1 OVERFLOW
        JNB P2.0,S4                     ;PERIOD COUNTER = 0
S5:     INC R1                          ;END OF MEASUREMENT
        JB P2.0,S5
        CLR TCON.4
        MOV A,R2                        ;STOP TIMERO
        ADD A,R1                        ;CALCULATE HIGH_COUNTER
        MOV R2,A
        MOV A,R3
        ADDC A,#0
        MOV R3,A
        JMP THREEC
S6:     NOP
        JNB P2.0,S6
S7:     INC R1
        JB P2.0,S7
        JMP S3
THREEC: MOV A,R2
        CLR C
        RLC A
        MOV R4,A                        ;R4 = 2*R2 = 2*HIGH_COUNTER_LOW_BYTE
        MOV A,R3
        RLC A
        MOV R5,A                        ;R5 = 2*R3 = 2*      ,,       HIGH_BYTE
        MOV A,R4
        ADD A,R2
        MOV R4,A                        ;R4 = 3*HIGH_COUNTER_LOW_BYTE
        MOV A,R5
        ADDC A,R3
        MOV R3,A                        ;R3 = 3*HIGH_COUNTER_HIGH_BYTE
        MOV R2,TLO                      ;CORRECT OFFSET
        CLR C
        MOV A,THO
        SUBB A,#08H
        MOV R1,A
        RET
```

```
;*********************************************************************************
FAST:
;
;                    ZET RESULTATEN UIT DE TWEE TIMERS IN R1_R2 (T1) EN
;                    R3_R4 (T0)
;*********************************************************************************
RSEG ROM
         MOV IE,#0
         MOV TCON,#0                  ;INITIALISING TIMERS
         MOV TL0,#0
         MOV TH0,#0
         MOV TL1,#0
         MOV TH1,#08H
         MOV TMOD,#019H               ;OFFSET IN TIMER1
K1:      JB P3.2,K1
K2:      JNB P3.2,K2                  ;SYNCHRONISING
         MOV IE,#081H
         MOV 87H,#01H                 ;INTERRUPT ENABLE
         SETB TCON.4                  ;INVOKE IDLE
         SETB TCON.6                  ;TIMERS IN RUN MODE
         CLR IE.7
         JNB P3.2,K3
         JNB TCON.7,K4
         MOV IE,#081H                 ;TEST ON TIMER1 OVERFLOW
         MOV 87H,#01H                 ;INTERRUPT ENABCE
         CLR TCON.4                   ;INVOKE IDLE
         CLR TCON.6
         CLR IE.7
         JMP VRWRK
K4:      JB P3.2,K4
         JMP K3
VRWRK:   MOV R2,TL1
         CLR C                        ;CORRECT OFFSET
         MOV A,TH1                    ;NOW IT WILL FIT IN 16 BITS AGAIN
         SUBB A,#08H
         MOV R1,A
         MOV R3,TH0
         MOV R4,TL0
         RET
```

$$\text{duty-cycle} = \frac{3*\text{HIGH\_COUNTER}}{\text{TIMER0}}$$

Normally HIGH_COUNTER is a 16 bits result which will be stored in two 8 bit registers. This causes a decreasing sampling rate because 2 extra commands are needed to glue these registers (test on overflow of the low_byte and depending on this test incrementing of the high_byte).
This problem can be solved when some calculations are done while the input signal is low. Then HIGH_COUNTER is waiting until the signal goes high again. This time can be used to "calculate" HIGH_COUNTER (figure 2).
What we see is a temporary result register, temporary_high_counter, which contains the number of samples while the input signal was high (of one period). As soon as the input signal goes low, the value of HIGH_COUNTER is calculated by adding the temporary_high_counter. For that moment the signal is not sampled. This constricts the duty-cycle to a limit. When the period is 300μs while the calculations take 15μs, then the duty-cycle mustn't exceed 0.95. Practically the duty-cycle never exceeds 0.9 (130°C) so this will never be a problem.

The measurement time is kept in hardware timer/counter TIMER0. It is started and stopped by software at a 1-0 transition of the input signal. In that case temporary_high_counter doesn't have to count so this action is of no influence on the sample rate.
The speed of incrementing the temporary_high_counter (the sampling rate) is 3μs.

C. Examples

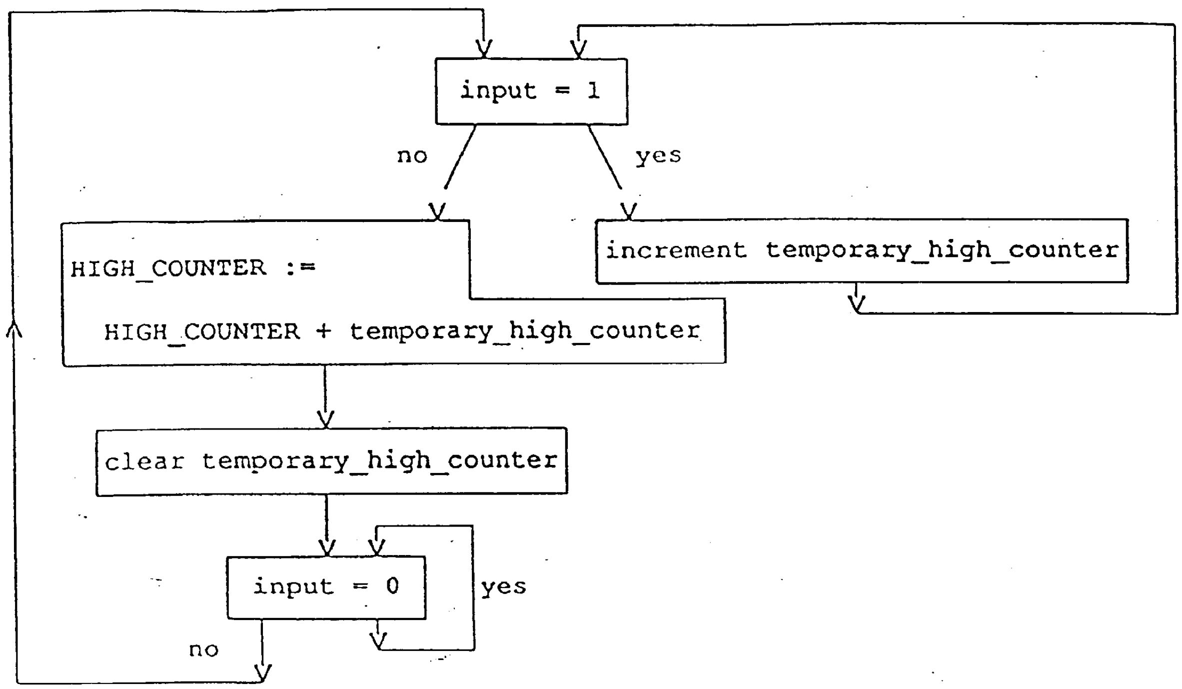
The sampling noise σ, of a duty-cycle modulated signal is calculated by:

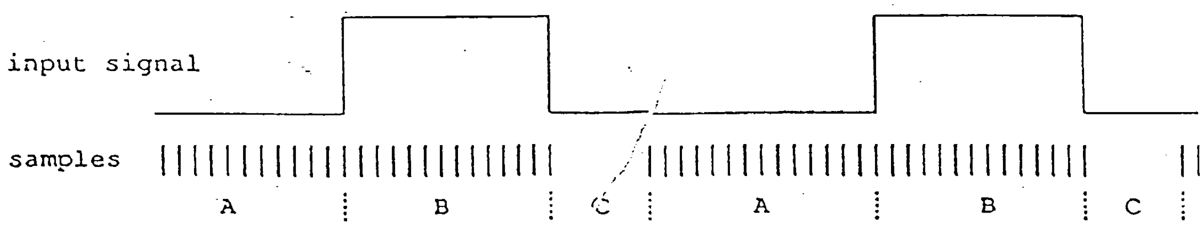
$$\sigma = \frac{t_s}{\sqrt{6T_m \cdot T_P}} = \frac{1}{\sqrt{6N'}} * \frac{t_s}{T_P}$$

where:
t_s = time between successive samples
T_P = period of the input signal
T_m = measurement time ($=N*T_P$)
N = number of periods within 1 measurement

The period of the input signal is between 300μs (40°C) and 800μs (-40 or 120°C).
The measurement time is about 64ms (a little smaller then $2^{16} \cdot 1\mu s$ because of the offset).
When the sampling rate is 1μs then the sampling noise is between $\sigma \cong 5 \cdot 10^{-5}$ and 10^{-4}. As a rule of thumb we can say that 95% of all values are read in the range of $\pm 2\sigma$ around the mean value (Gaussian distribution).

When the sampling rate is 3μs the sampling noise is 3 times more: $\sigma \cong 1.5\text{-}3 \cdot 10^{-4}$.





A: waiting for the input signal to go high again.

B: incrementing temporary_high_counter

C: calculating HIGH_COUNTER followed by clearing temporary_high_counter

Figure 2 a) Flow diagram of a duty-cycle measurement by software (only the part to measure the "high"-time).

b) Time diagram belonging to figure 2b.