Measuring Duty Cycles with an Intel MCS-51 Microcontroller

Paul C. de Jong and Ferry N. Toth

The fastest way of measuring duty cycles is with the aid of hardware. The MCS-51 type of microcontrollers offers possibilities for that since they are equipped with two internal timer/ counters.
The two port-pins INT0 (P3.2) and INT1 (P3.3) can control these timer/ counters directly by hardware. Therefore we consider them as “fast-inputs”. All other pins can control the timer/ counters only by software. The internal configuration of the hardware including the smart sensor SMT 160 is shown in Fig. 1.

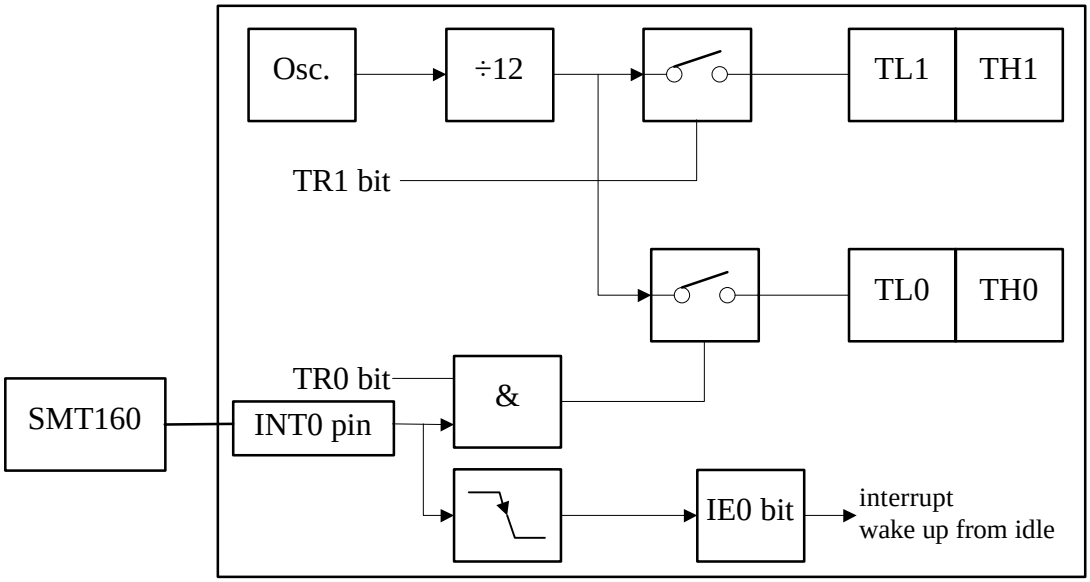

Figure 1. Configuration of internal 8051 hardware

This application note describes A) four assembly programs for the measurement of duty-cycles by hardware and B) a program for the measurement by software. Finally, we will discuss the conditions for which averaging can improve the resolution.

A1. Hardware-controlled Measurement via INT0 pin

When the duty-cycle is measured by hardware there are some restrictions concerning the tasks the CPU is performing: Both timers 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 occurs when using this fastest and most accurate way of measuring, which is obtained by using interrupts preceded by the IDLE-mode of the CPU. This is important because, when the processing of an instruction would be interrupted, the instruction is first completed and not all instructions have an equal execution time. During the IDLE-mode, the CPU is non-active. The two timers are insensible to the IDLE command which is an important feature of the MCS-51. 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 to operate in the 16-bits timer mode. Therefore the “timer/ counter mode control” (TMOD) register is initialised with the value 19H. In this mode TIMER0 only runs when P3.2 is logically “1”, while TIMER1 can only be controlled by software. TIMER1 measures the total measurement time. After a measurement the duty-cycle *p* is obtained by:

$$p = \frac{contents\_of\_TIMER0}{contents\_of\_TIMER1}$$

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).
The measurement is explained with the aid of a flow diagram (fig. 2). Firstly, the contents of both timers are set. The initialising 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, which occurs at the next 1-0 transition of the input signal, the flags TR0 and TR1 in the “timer control” register (TCON) 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 continuously running during the whole measurement. Note that TIMER1 starts 3μs too late because processing of an interrupt takes 3μs. However because the measurement will be stopped in the same way this delay is eliminated in the final result.
With respect to the measurement time, there is the choice to fix either the number of periods or the measurement time. Because the period duration can vary between 300μs and 800μs a fixed number of periods is an inefficient option. For short periods the measurement time is also short so the result will be troubled because of sampling noise. Therefore a fixed measurement time is chosen corresponding to 16 bits of machine cycles. Now the measurement will be finished after TIMER1 generates an overflow. In this case we have a 17 bits result which would require complex software routines. This problem is solved when TIMER1 is initialised (before the measurement) with an offset. This offset corresponds to the maximum length of two periods of the sensor signal (about 1.6ms in time). The offset is subtracted from the 17 bits result and will results in a 16 bits word.
When TIMER1 generates an overflow the measurement has to be stopped (see the right-hand branch in figure 2). After occurrence of the next 1-0 transition of the sensor signal the interrupt-enable flag is set and once again the IDLE mode is invoked. After occurrence of the interrupt the flags TR0 and TR1 in the TCON register are cleared.
After correction for the offset in TIMER1, the contents of both timers are used to calculate the duty-cycle.

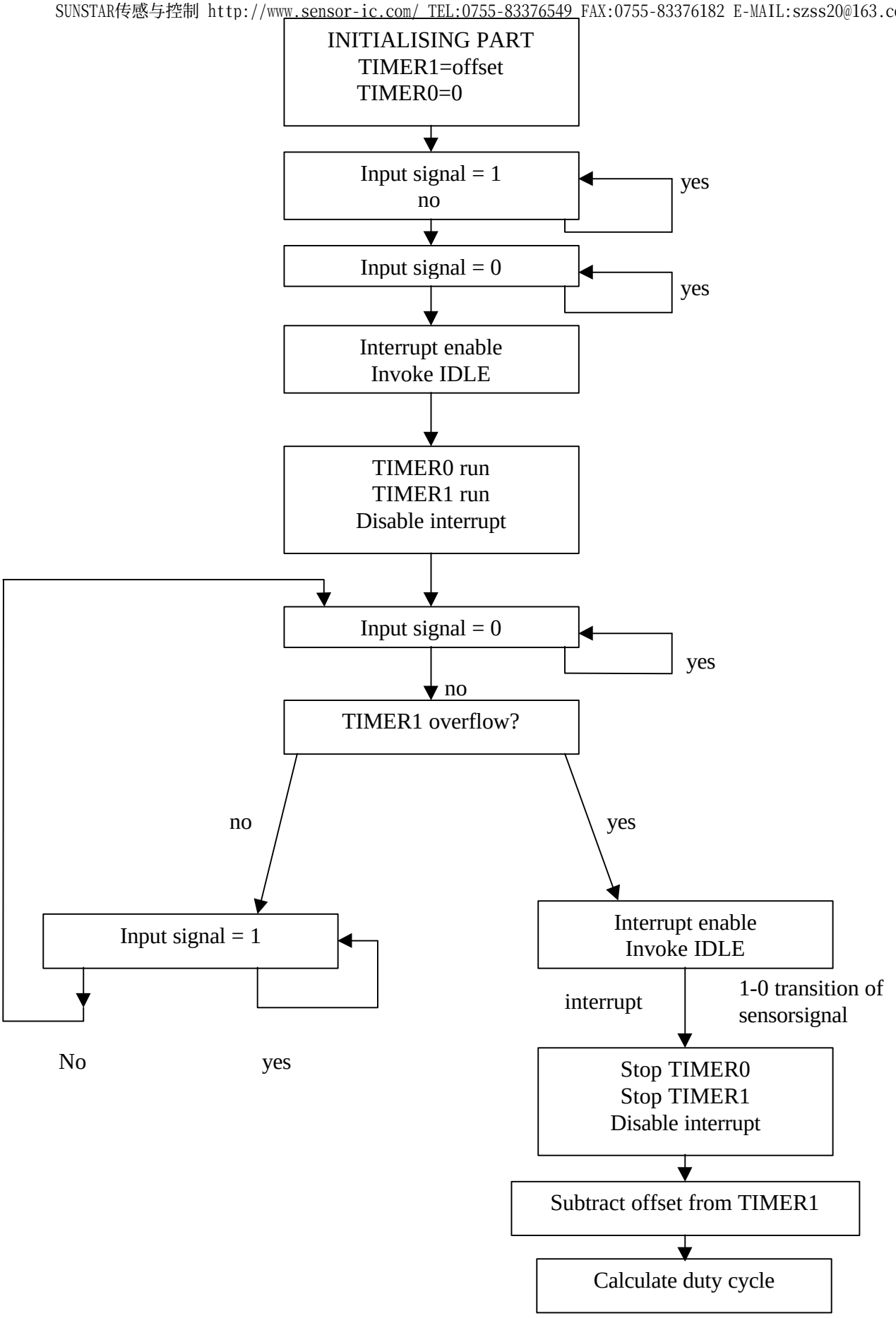

Figure 2. Flow diagram of a duty-cycle measurement with a resolution of one machine cycle.

A2. Measurement via INT0 pin with serial communication

The method proposed in the previous section uses the 8051 IDLE mode to create a constant delay (interrupt latency) between the moment of interrupt (on the falling edge of the input signal) and the moment of sampling Timer0. This is necessary because when the processing of an instruction is interrupted, the instruction is first completed and not all instructions have an equal execution time.

During the IDLE mode no instructions are being processed because the execution unit of the 8051 is disabled. However, the interrupt timer and serial units are left running. In this mode the power consumption is significantly reduced.

To realize a constant latency only one interrupt source may be enabled (as is the case in Figure 1). However, in some cases we require simultaneous temperature measurement and serial communication. The serial communication is likely to require and interrupt of its own as well as a timer to generate the baud rate. Using a 33MHz 8051 it is possible to realize 1200 baud communication and duty-cycle measurement, without additional hardware, as shown in Fig.3.

In this case Timer 1's overflow rate is required to generate the baud rate. Using a 32.9856 MHz crystal Timer 1 needs to count 859 clocks to overflow. proposed that it is used in a 16 bit mode. The Timer 1 overflow bit generates an interrupt (TF1) to reload the divider value. Since we are using a fast processor this same interrupt handler can increment a software counter in a short time. This software counter combined with the actual value of Timer 1 is used as a time base to determine the period of the SMT160 output signal. Timer 0 retains its function for counting the high period of the signal.

Of course, since we are now using 3 interrupt handlers (INT0, TF1, Serial) the interrupt latency is not constant anymore, so the resolution of the measurement will be degraded by a factor of 3. However, this is compensated by the higher clock speed of the processor.

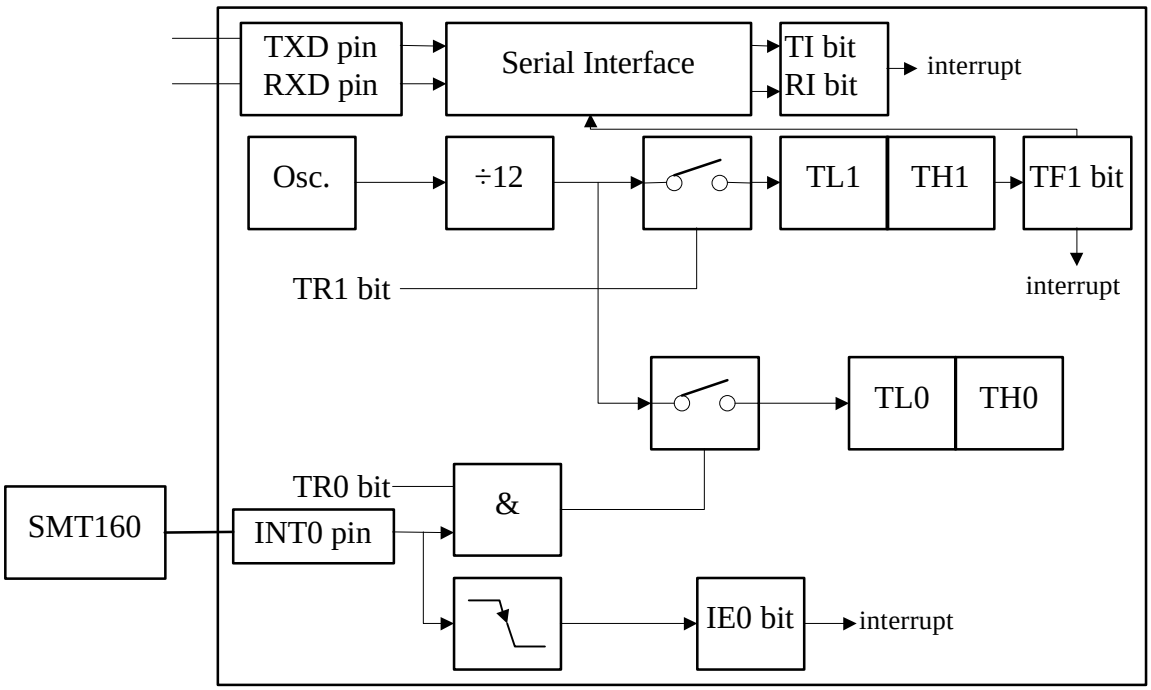

Figure 3. Simultaneous measurement and serial communication

A3. Duty-cycle measurement using Timer 2; capture register

Many 8051 derivatives, including 8052 and the 16 bit 8051XA, are equipped with an additional timer, Timer 2 (Fig. 4). Timer 2 is an advanced 16 bit timer/counter with capture/reload register. In our case, the function of the capture register is to instantaneously load the value of Timer 2 (capture) and hold it until the interrupt handler reads it. This eliminates the effect of the interrupt latency, provided that the latency is less then the interrupt rate.

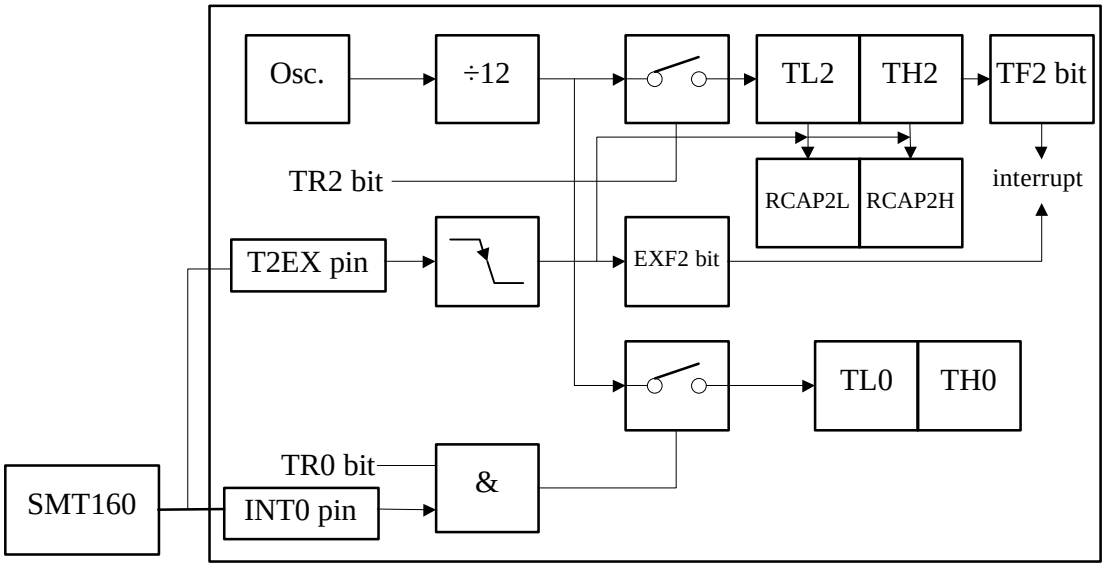

Figure 4. Duty-cycle measurement with Timer 2 and Timer 0.

Using Timer 2 for the measurement of the period of the SMT160 signal and Timer 0 for the high period, Timer 1 is free to be used as a baud rate generator for the serial interface. Sometimes, Timer 0 can not be spared for the measurement of the high period of the SMT160 signal, for instance when real time operating system (RTOS) is used. The scheduler of the RTOS often requires a clock to generate the time slices of each process.
In that case Timer 0 might be in use by the RTOS. By adding an external XOR gate (Fig. 5), Timer 2 will be sufficient to measure the duty-cycle of the SMT160. By toggling pin OUT in the interrupt handler of Timer 2, both rising and falling edges can be captured.

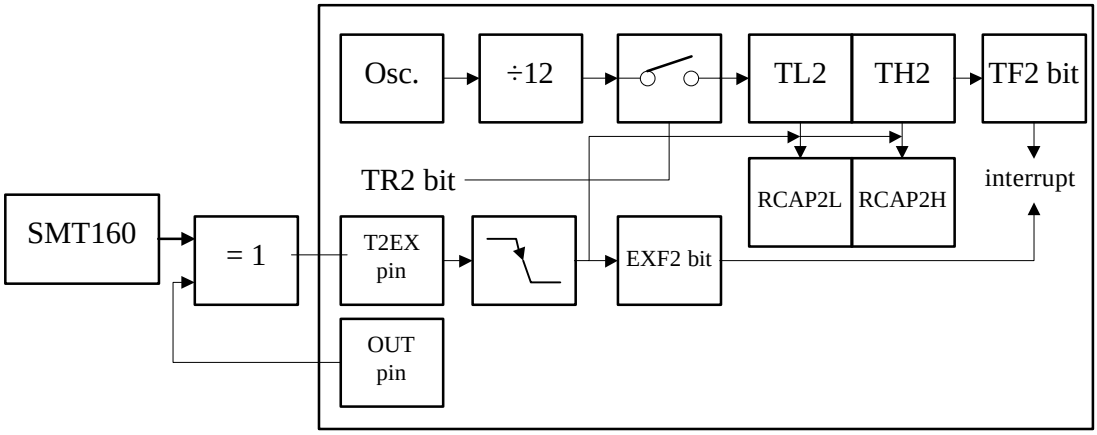

Figure 5. Duty-cycle measurement using Timer 2 only.

A4. Duty-cycle measurement using a programmable counter array (PCA)

The 8051FX derivatives are equipped with an additional piece of hardware: the programmable counter array. This consists of one timer and 5 capture registers.
The timer can be programmed to run at a frequency Osc/12 or Osc/4. As compared to the ordinary 8051, the use of the FX types enables to perform the measurements with 3 times the resolution in the same measurement time.

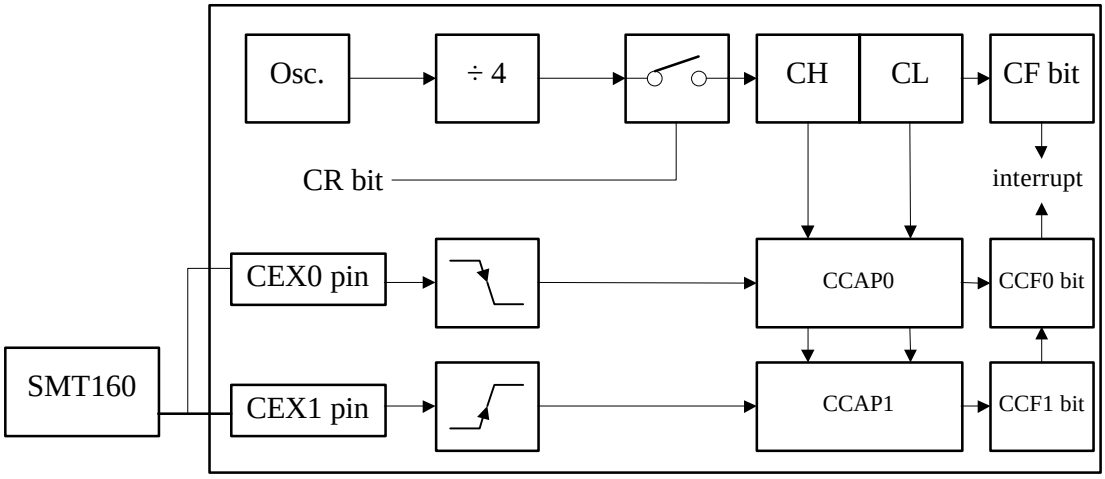

Figure 6. Duty-cycle measurement using the PCA.

Moreover, the capture registers can be programmed to capture on rising or falling edges, or both, so no external XOR gates are required. Since there is a capture register available for both the rising and the falling edge, interrupt latencies are non critical using this processor family (Fig. 6). This means the interrupts handlers can be easily written using a high level programming language like C.
A example of an interrupt handler that measures *n* periods of the SMT160 signal consecutively is given below.

```
void PCAHandler(void) interrupt 6 using 1 {
   static union Word2Byte CaptureUp, CaptureDo;

   if (PCAOverFlow) {                                  /* PCA Overflow? */
      PCAOverFlow = FALSE;
      if (!Ready) {
         if (OverFlow > 3) {                    /* 3 overflows => error */
            SetCaptureOff();                           /* Capture off */
            PCACapture0 = FALSE;                       /* Clear flags */
            PCACapture1 = FALSE;
            Ready = TRUE;                          /* measurement done */
                                                          Error = TRUE;
         } else {
            OverFlow++;
         };
      };
   } else {
      if (PCACapture1) {                                /* rising edge? */
         PCACapture1 = FALSE;                             /* Clear flag */
         if (!Ready) {
            CaptureDo.Byte.Hi = CCAP1H;               /* save PCA value */
            CaptureDo.Byte.Lo = CCAP1L;
            HiTime += (CaptureDo.Word - CaptureUp.Word);
         };                                      /* determine low period */
```

```
      } else {
         PCACapture0 = FALSE;
         if (!Ready) {
            CaptureUp.Byte.Hi = CCAP0H;   /* save PCA counter */
            CaptureUp.Byte.Lo = CCAP0L;
            if (First) {
               First = FALSE;          /* 1st time just caputure value */
                                                HiTime = LoTime = 0;
               SetPCA1NegEdge();              /* enable falling edges */
            } else {
               LoTime += (CaptureUp.Word - CaptureDo.Word);
               if (--Count == 0) {            /* when Count = 0 ready */
                  SetCaptureOff();                    /* capture off */
                  PCACapture0 = FALSE;                /* clear flags */
                  PCACapture1 = FALSE;
                  PCAOverFlow = FALSE;
                  Ready = TRUE;                  /* measurement ready */
               };
            };                                  /* determine high period */
         };
      };
      if (!Ready) {
      };
      OverFlow = 0;                              /* we have a signal */
   };
   return;
}
```

We interface with the interrupt handler from the main program, using the following functions:

```
#include <pca.h>
#include <stdio.h>
#define PERIODS 25

struct DoubleByte {
  unsigned char Hi, Lo;
};


union Word2Byte {
  unsigned short Word;
  struct DoubleByte Byte;
};


static volatile bit First, Ready, Error;
static volatile unsigned int Count = 0, Periods = 51;
static volatile unsigned char OverFlow;
static volatile unsigned long HiTime, LoTime;

void StartCount(void) {
   First = TRUE;                         /* Initializes all varialbes */
   OverFlow = 0;
   Count = Periods;
   Ready = FALSE;
   Error = FALSE;
   SetPCA0PosEdge();                                /* Enable capture */
}

void SetPeriods(unsigned APeriods) {
   Periods = APeriods;
}

bit IsReady(void) {
   return(Ready);
}

bit IsError(void) {
```

```
   return(Error);
}

float GetDutyCycle(void) {
   return (float)HiTime / (HiTime + LoTime);
}
```

The main program needs to initialize the interrupt handler once, the repeatedly start a measurement, wait till the measurement is finished, check for errors and display the result.

```
main() {
   SetPeriods(51);
   while(TRUE) {
      StartCount();
      while(!IsReady()); continue;
      if(IsError()) printf("An error has occured\n");
      else printf("The termperature is %f\n",
         (GetDutycycle() - 0.32) / 0.0047));
   }
}
```

B. Software-controlled Measurement

Only the I/ O ports P3.2 and P3.3 can be used to detect interrupts. Therefore, when sensors are connected to the other I/ O ports only a software-controlled measurement can be used to measure the duty-cycle. Again two counters are required. However, it is still possible to use a hardware timer, although it is software controlled. This timer TIMER0 can count the measurement time. A fast software routine is used to measure the "1" state of the sensor signal. The results are stored in a counter called: HIGH_COUNTER.
The timer TIMER0 increments every machine cycle, which takes 1μs. The software sample rate takes 3μs. Therefore, to obtain the duty-cycle *p*, HIGH_COUNTER is multiplied by 3, according to the equation:

$$p = \frac{3 \times HIGH\_COUNTER}{TIMER0}$$

Normally HIGH_COUNTER should store more than 8 bits and therefore requires two 8-bits registers. This would cause a decrease of the sampling rate, because two extra commands would be needed to "glue" these registers (test on overflow of the low_byte and, depending on the test result, incrementing of the high_byte). Therefore, an alternative solution has been applied: When the input signal is low, HIGH_COUNTER is waiting until the signal goes high again. This time can be used to "calculate" HIGH_COUNTER (figure 7). This figure shows the use of a temporary-result register which is called : temporary_high_counter. This counter contains the number of samples for which the input signal was high during one period. As soon as the input signal goes low, the value of HIGH_COUNTER is calculated by adding the temporary_high_counter to it During this calculation interval the sensor signal is not sampled. This restricts the duty-cycle to a limit. However the calculations take only 15μs, so that even when the duty-cycle equals 0.95 for a period of about 600 μs, there will not be a problem. The counts for the measurement time are stored in the hardware timer/ counter TIMER0. It is started and stopped by software at a 1-0 transition of the input signal. In that case

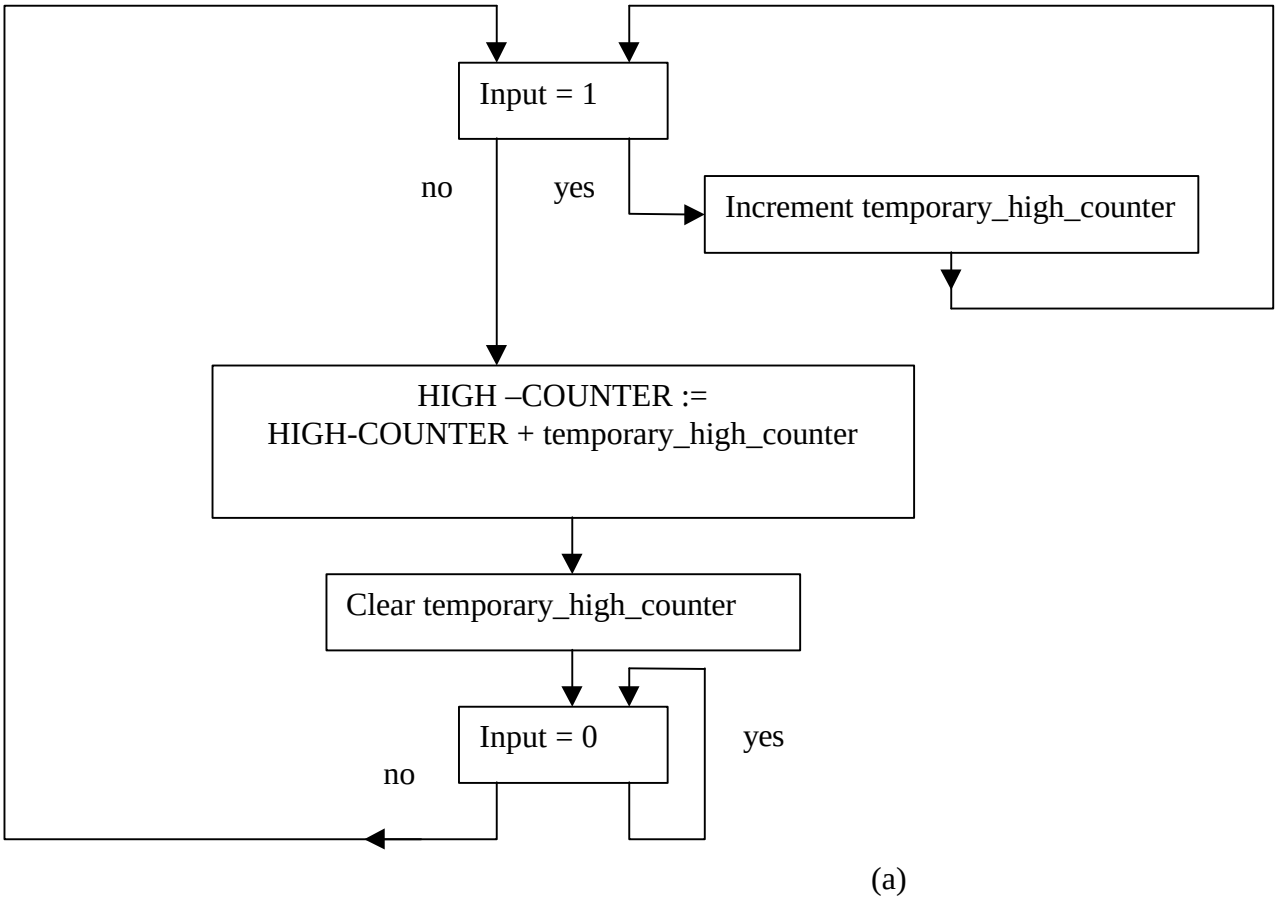

Figure 7 a) Flow diagram of a duty-cycle measurement by software (only the part to measure the "high"-time).

b) Time diagram belonging to figure 7a. During interval C HIGH_COUNTER is calculated followed by clearing of the temporary_high_counter.

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.

Examples: The standard deviation σ of the sampling noise, of a duty-cycle modulated signal can be calculated from the equation:

$$s = \frac{t_s}{\sqrt{6T_m \times T_p}} = \frac{1}{\sqrt{6N}} \times \frac{t_s}{T_p},$$

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

The period of the input signal is between 300µs (at 40^0C) and 800µs (at -40^0C or 120^0C). The measurement time is about 64ms (slightly smaller than 2^{16} × 1µs because of the offset). When the sampling rate is 1 µs, then the sampling noise is between ó≅ 5× 10^{-5} and 10^{-4}. As a rule of thumb we can say that 95% of all values are read in the range of ±2σ around the mean value (Gaussian distribution).
When the sampling rate is 3µs the sampling noise is 3 times more:

C. Understanding averaging of measurement results

Using an ordinary A/D converter averaging successive measurements will not yield an improved resolution. With duty-cycle measurements the resolution can be improved by averaging successive measurement results - under certain conditions.
Obviously the successive measurement results should not be correlated. This is true when the period of the period SMT160 is not an integer multiple of the period of the microcontroller's timer. We can ensure this by measuring the jitter of the falling edges using a frequency counter, with the gate time set to τ ms. This jitter appears to be a function of τ. This effect can also be visualized on an oscilloscope with DTB function. This function will allow you to zoom in on the n^{th} falling edge after the trigger. As you can see, the jitter increases with n. When this jitter is larger then one period of the microcontroller's timer, successive duty-cycle measurements will not be correlated. This means the resolution will increase with the square root of the number of samples (as described elsewhere), when a certain minimum delay between successive measurements is observed.

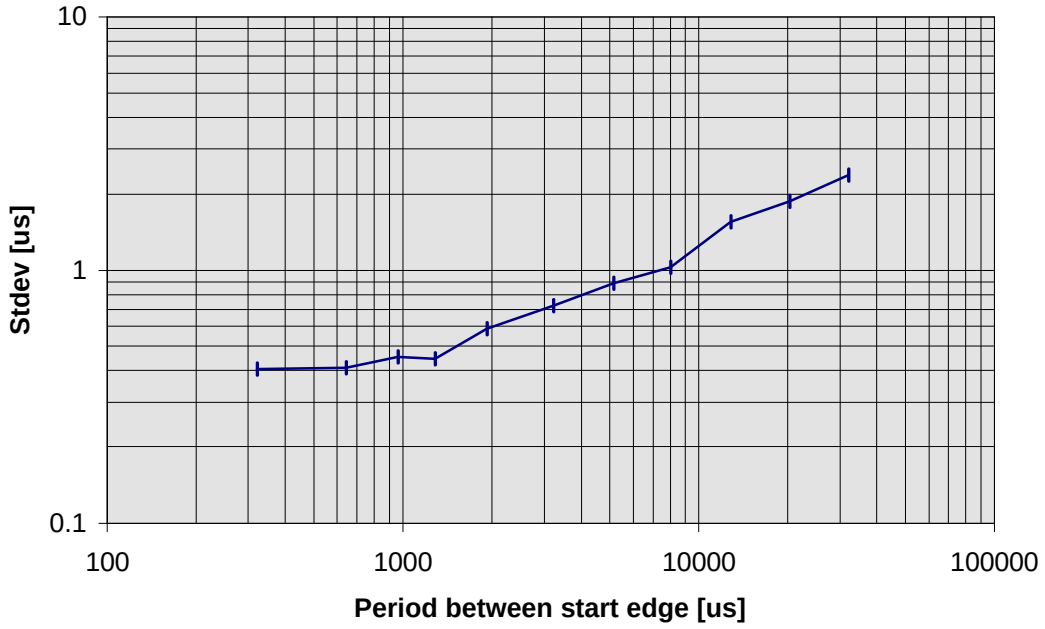

Figure 8. Jitter as a function of the gate time

The above figure was actually measured using a microcontroller with 1.25 MHz clock. Using this setup, the quantization noise approximately equals the thermal (and other) noise with a 1 ms interval between measurements. This means that for uncorrelated duty-cycle measurements a 1 ms interval must be observed. In another setup the noise might show different behavior due to electromagnetic interference etc.
From the figure it can also be estimated that with a 4 MHz clock (for instance a 8051FA running at 16 MHz and using the PCA) a zero delay between measurements can be used, thus obtaining maximum measurement speed.