Contact information
Headquarters:
Sciosense B.V.
High Tech Campus 10
5656 AE Eindhoven
The Netherlands
info@sciosense.com
www.sciosense.com

acam-messelectronic gmbH

is now

Member of the ams Group

The technical content of this acam-messelectronic document is still valid.

Contact information:

Headquarters:
ams AG
Tobelbaderstrasse 30
8141 Premstaetten, Austria
Tel: +43 (0) 3136 500 0
e-Mail: ams_sales@ams.com

Please visit our website at **www.ams.com**

Ultrasonic Flow Converters

Application Note

Ultrasonic Water & Heat Metering with TDC-GP22

July 10th, 2013, Version 1.2

Document No.: AN032_en.pdf

Published by acam-messelectronic gmbh
©acam-messelectronic gmbh 2013

Disclaimer / Notes

Support / Contact

For a complete listing of Direct Sales, Distributor and Sales Representative contacts, visit the acam web site at:

http://www.acam.de/sales/distributors/

For technical support you can contact the acam support team in the headquarters in Germany or the Distributor in your country. The contact details of acam in Germany are:

support@acam.de or by phone +49-7244-74190.

mess•electronic

Content

1 Overview

In 2012 acam launched the TDC-GP22 Time-to-Digital Converter. It comes with extended functionality and provides great benefit to designers that demand for ps resolution in time of flight applications, e. g. like ultrasonic water and heat meters.

Especially the integration of analog elements like a offset stabilized comparator and analog switches simplifies the hardware design and reduces the number of external components to a minimum.

This application note is intended as an add-on to the TDC-GP22 datasheet and describes the implementation of an ultrasonic heat meter front-end with the TDC-GP22. It provides additional information that helps to decrease design time and avoid circuit problems due to wrong component values or incorrect configuration of the device. We recommend to follow the layout guidelines and to copy the schematics of the demo board. This is well proven and shows best performance with respect to offset drift and noise. Finally, the application note contains a generic software example that shows a typical measurement flow.

2 Introduction

This application note describes the implementation of an ultrasonic heat meter front-end that is based on acam's TDC-GP22 Time-to-Digital Converter. The GP22 operates as a front-end device that provides fully automated control of the flow- and temperature measurement sequence. The amount of heat dissipated is calculated from the flow rate and the temperature difference of the incoming and outgoing fluid by an external micro-controller.

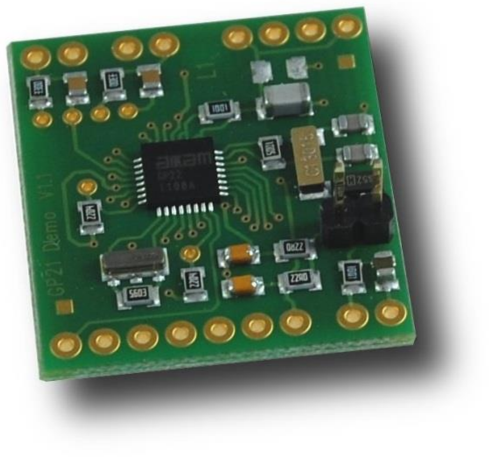


Figure 1.1: GP22 Front-end electronics for heat meters

Design Goals:

- Compact and cost saving hardware design that keeps external components to a minimum
- Low current consumption (down to 2 µA for a complete measurement cycle including the analog part, transducers and PT sensors)
- High resolution time-of-flight measurement that allows compact and small spool pieces with small diameters
- High resolution but low current temperature measurement
- Open system architecture offers a high degree of flexibility with an external microcontroller and a modular system design

2.1 Flow Measurement

The ultrasonic flow measurement is based on the transit-time principle. Two transducers are connected to the TDC-GP22. They can both operate as an ultrasonic transmitter or receiver.

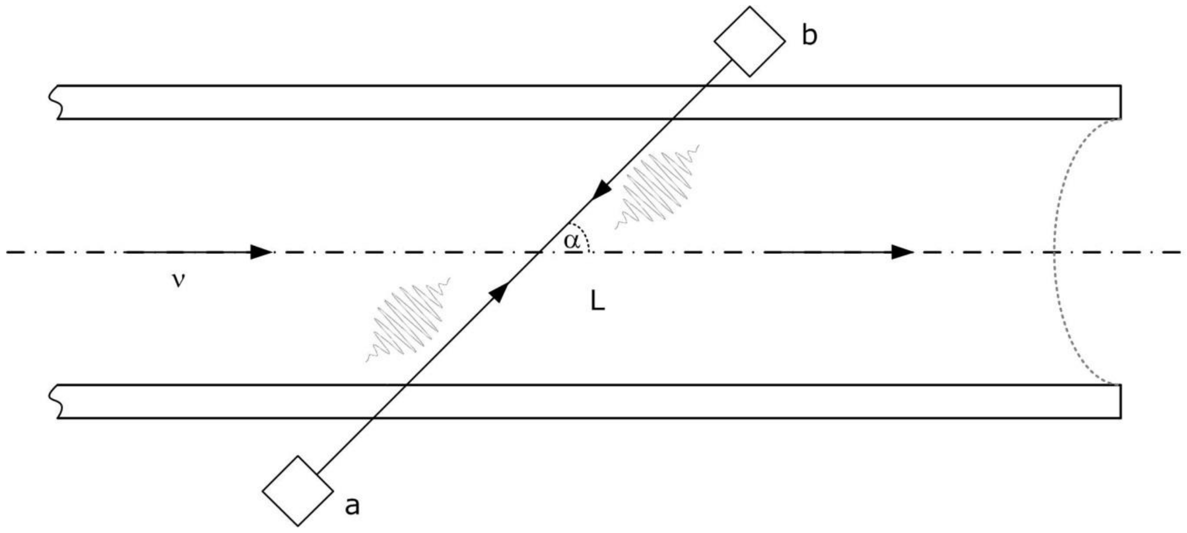


Figure 1.2 Basic principle of ultrasonic flow metering by means of transit time difference measurement

The GP22 alternately transmits a 1 MHz signal burst (20 pulses) between the two transducers and measures the transit time that a pulse burst takes for traveling from transducer a to b and the opposite direction from b to a. The transit time of the ultrasonic signal propagating in flow direction is shorter than the transit time of the signal propagating against flow direction.

For a given average fluid velocity v, a measurement path L and a sound velocity C_0 in the fluid, an acoustic signal needs the time t_{up} for the upstream path and t_{down} in downstream direction. They can be calculated as follows:

$$t_{up} = \frac{L}{c_0 + v * \cos \propto}$$

$$t_{down} = \frac{L}{c_0 - v * \cos \propto}$$

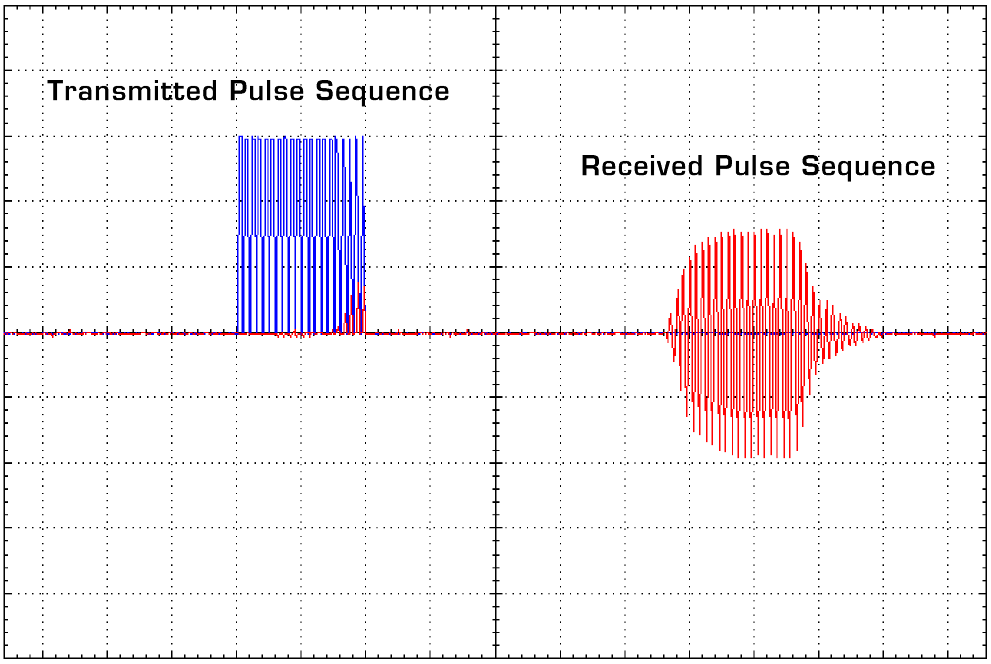


Figure 1.3: Send- / received pulse sequence
(Sent pulse=1V/Div.; Received pulse=200mV/Div.; Timebase=10µs)

The computation of the result is then performed by an external microcontroller. In a first step it calculates the time difference of the delay times in up- and down direction and after that the velocity v.

$$\Delta t = t_{down} - t_{up} = \frac{2 * L * v * \cos \propto}{{c_0}^2 - v^2 * \cos^2 \propto} \approx \frac{2 * L * \cos \propto}{c_0^2} * v$$

$$v \approx \frac{\Delta t * {c_0}^2}{2 * L * \cos \propto}$$

2.2 Temperature Difference Measurement

For heat meters it is important to measure the temperature with very high resolution but low power consumption on the other hand. The TDC-GP22 uses the PICOSTRAIN method to measure the temperature difference of the incoming and outgoing water. It transfers the temperature dependent resistance variation of an RTD in an accurate time interval measurement. A capacitor is discharged alternately through the RTD sensor and a temperature stable reference resistor and the discharge times are measured with picosecond resolution. The Figure below shows a signal curve for a temperature measurement sequence with seven additional Fake[1] measurements.

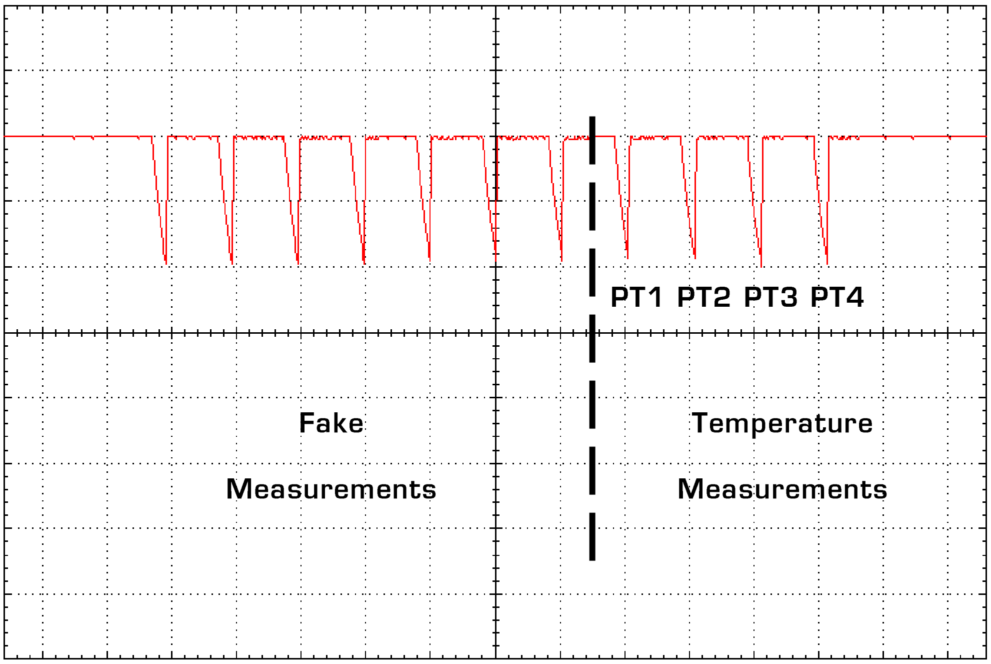


Figure 1.4: Oscillogram of the fake- and temperature measurement charging-/discharging cycles, measured at R9 (Cycles=1V/Div.; Timebase=500µs)

The result is then provided in the corresponding GP22 result register 0 to 3. The ratio of the discharge time shows the deviation of the sensor resistance relative to the reference resistor. From the given results T_{RTD} and T_{Ref} the external microcontroller calculates the temperature difference as follows:

$$R_{RTD} = \frac{\left(\frac{T_{RTD}}{T_{Ref}} - 1\right) * R_o}{gainfactor}$$

For the internal comparator at 3.0 V the gain correction factor is 0.9931. Now the temperature can be calculated by solving the RTD transfer function for the temperature:

$$\vartheta = \frac{-R_o * A + \sqrt{(R_0 * A)^2 - 4 * R_0 * B * (R_0 - R)}}{2 * R_0 * B}$$

[1]Fake measurements improve stability of the temperature measurement by adding additional charging /discharging cycles before the actual temperature measurement starts

Finally, the temperature difference is calculated as follows:

$$\Delta\vartheta = \vartheta_{RTD} - \vartheta_{Ref}$$

3 Circuit Description

The complete schematics and the layout can be found in section 3 of this document. To get best results and to avoid circuit problems we recommend to use it as a reference for your design. The main considerable design items are now described in the following chapters.

3.1 Power Supply Considerations

The TDC-GP22 needs two supply voltages for chip internal power supply. The core supply voltage Vcc and the I/O supply voltages Vio. Both can be generated from one and the same source. Usually, a linear regulator. Here, the XC6206 from Torex Semiconductor is used. Because of its very low quiescent current of only 1 µA in operation it is ideal for power critical applications, like battery driven heat or flow meters. Additionally, a good de-coupling and bypassing is recommended for noise reduction. The below schematics gives a reference for your design:

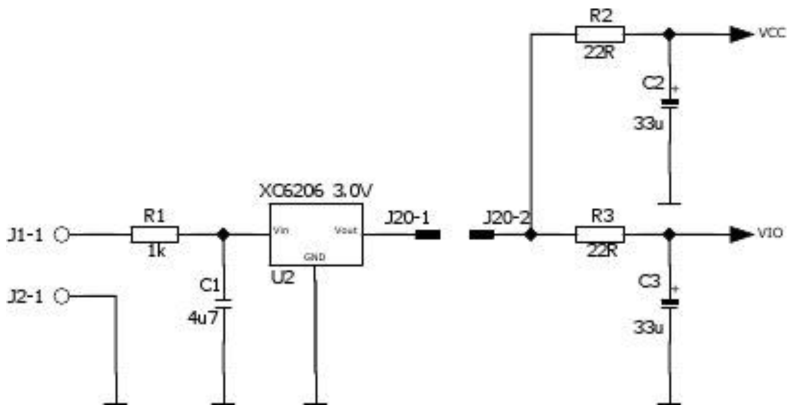


Figure 2.1 Power supply bypassing and decoupling

Important Note:

Do not use switched regulators in your power supply design. They would introduce a lot of noise that significantly reduces the measurement quality.

3.2 External Clock Sources

Battery driven applications like heat meters demand for very low current consumption. According to that the GP22 operates with two LVCMOS or LVTTL compatible clock sources. A 32.768 kHz quartz and a 4 MHz ceramic oscillator. For power efficiency the 4 MHz clock is controlled by the 32 kHz quartz and only switched on during time measurement.

3.2.1 The 32,768 Hz Quartz

The 32.768 kHz quartz is responsible for start-up control of the 4 MHz clock and is used as reference for calibrating the 4 MHz ceramic oscillator, as explained in chapter 2.2.2. The 10 pF capacitors together with the paralleled 10M resistor guarantuees stable oscillation.

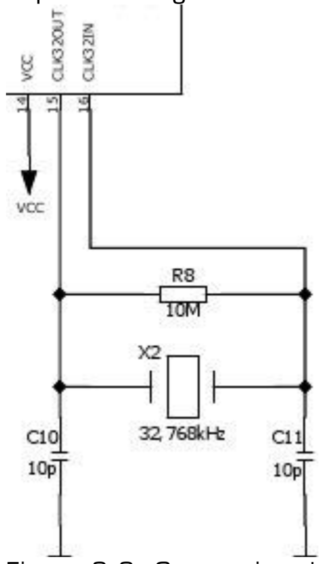


Figure 2.2: Connecting the 32.768 kHz quartz oscillator

3.2.2 The 4 MHz High Speed Clock Oscillator

As 4 MHz clock source a standard ceramic oscillator in combination with a 560 k parallel resistor can be used. The high speed clock circuit is shown below.

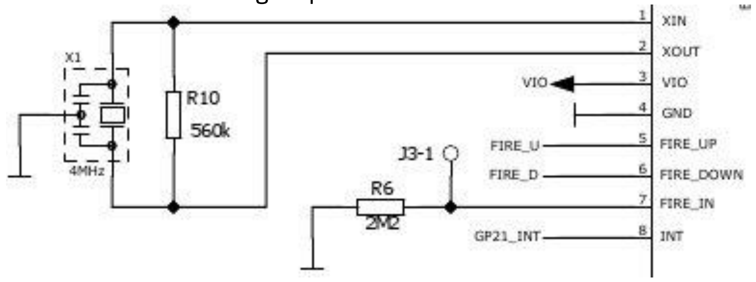


Figure 2.3 Circuit diagram for the 4 MHz ceramic resonator

Compared to a crystal oscillator it is more cost efficient and thanks to the short settling time it can be operated in a power saving switched mode. Then the active states are controlled by the GP22 internally.

The GP22 integrated clock calibration helps to overcome the poor accuracy and temperature drift behavior that is typical for a ceramic oscillator. This offers quartz accurate time measurement, even when a ceramic oscillator is used as high speed clock. For more details about how to use the clock calibration please have a look to section 5.1.4 of the TDC-GP22 datasheet.

3.2.3 Clock Option for external Microcontroller

To save external components the GP22 can provide a 4 kHz or a 32 kHz signal, e. g. as a system clock for the external microcontroller. The 32 kHz can be provided through EN_START pin and has to be configured by SEL_TSTO2 parameter in configuration register 1. A 4 kHz signal is available on FIRE_IN by appropriate configuration of the
SEL_TSTO1 parameter in configuration register 1.

3.3 Transmitter / Receiver Circuit

Basically, the transmission circuit consists of two first order high pass filters. They are connected to Fire_Up and Fire_Down pins of the TDC-GP22 and their resistor outputs drive the piezo transducers. On the receiver side the capacitor outputs are connected to GP22 analog inputs. Here, COG capacitors should be used to achieve highest temperature stability. An integrated analog multiplexer operates the transducers alternatingly as transmitter / receiver. All elements are controlled by the TDC-GP22 and only powered up during measurement.

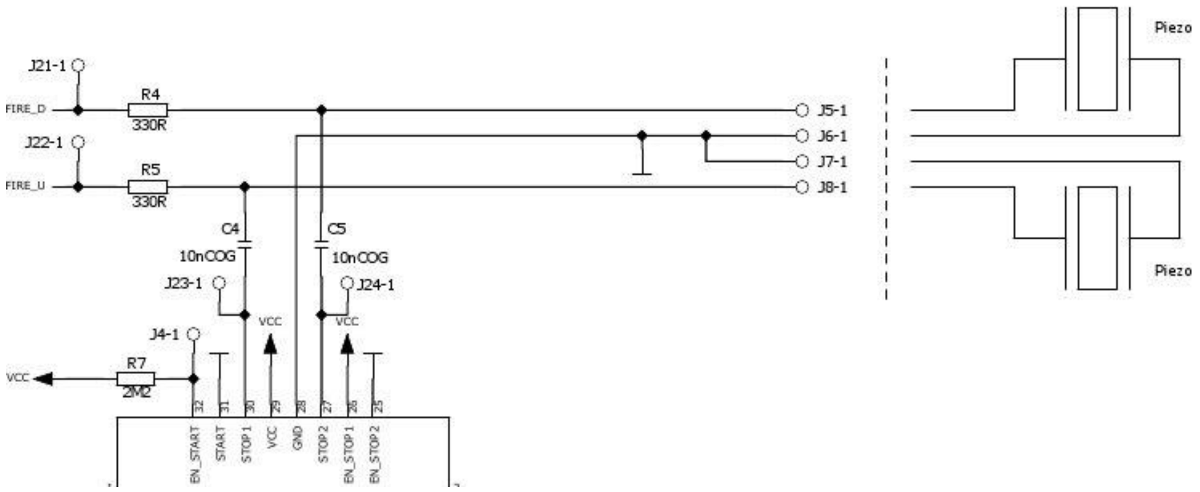


Figure 2.4 Circuit diagram of the transmitter / receiver signal path

As the integrated comparator cannot handle GND as threshold the LoadC capacitors (C4, C5) in the receiving path switch the signal to the comparator threshold of 1/3 Vcc. Then it is coupled to the TDC-GP2 analog inputs. Figure 2.5 shows the corresponding oscilloscope trace.

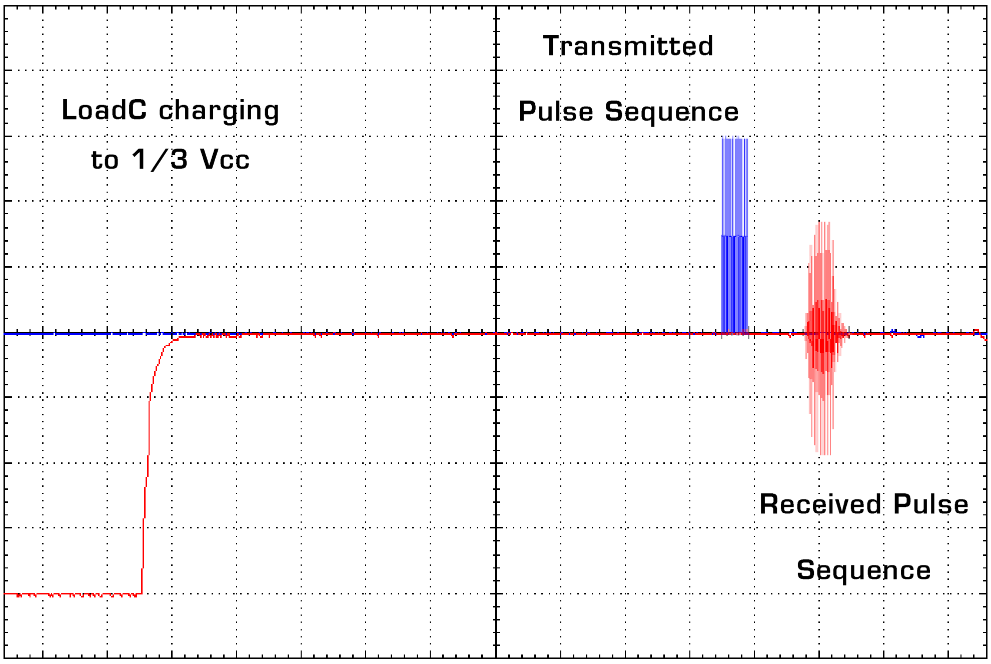


Figure 2.5: Signal curve on non-inverted comparator input
(Sent pulse=1V/Div.; Received pulse=200mV/Div.; Timebase=50µs)

Finally, the integrated low noise comparator generates a square wave signal from the received pulse sequence that provides the stop pulses for the GP22 internal time measurement unit.

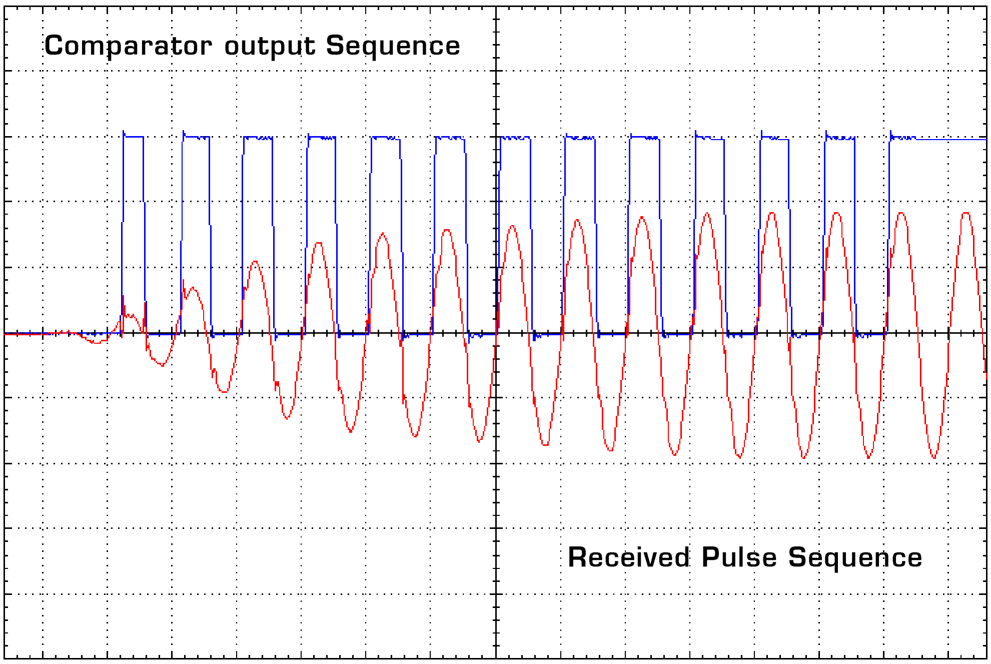


Figure 2.6: Received pulse sequence and corresponding comparator output signal
(Comparator output signal=1V/Div.; Received pulse=200mV/Div.; Timebase=1.0µs)

More details about the GP22 analog unit and how it operates can be found in section 4.3 of the TDC-GP22 datasheet.

3.3.1 Time Windowing by Stop Masking

The major improvement of TDC-GP22 is the implementation of the First Wave Mode.

The DELVAL1 stop masking is used to suppress any noise in advance to the acoustic received signal. The noise come e.g. from the fire pulses. The Figure below shows a typical configuration with DELVAL1 = 5000. This applies a time window with a delay of 39062,50ns delay (@ 4 MHz high speed clock) before the stop 1 channel is activated to detect the incoming stops. DELVAL2 and DELVAL 3 are not required and set to 0. The DELVAL1 setting should cover at least the duration of the fire burst (e.g. 20μs at 20 pulses with 1MHz). More details are described in section 4.2.3 of the TDC-GP22 datasheet.

The stop masking for the three time measurements is set by parameters DELREL1 to DELREL3, relative to the first wave. DELREL1 = 8, DELREL2 = 9, DELREL3 = 10 measure the 8th, 9th and 10th stop after the first wave. More details are described in section 4.4 of the TDC-GP22 datasheet.

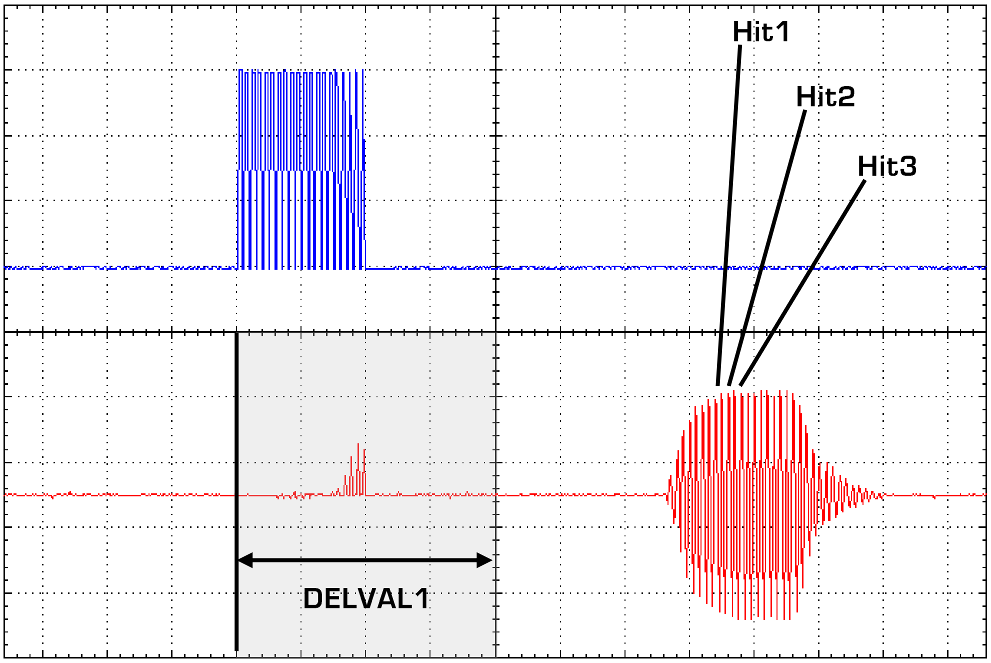


Figure 2.7: Sent and received pulse sequence
(Sent pulse=1V/Div.; Received pulse=200mV/Div.; Timebase=10μs)

3.4 Temperature measurement

A pair of PT 1000 sensors in combination with a temperature stable 1k reference resistor (R9, 50 ppm), is used for hot / cold temperature difference measurement.

A 100 nF COG type capacitor operates as load capacitor (C8). To minimize the number of external components the GP22 internal Schmitt-Trigger is used.

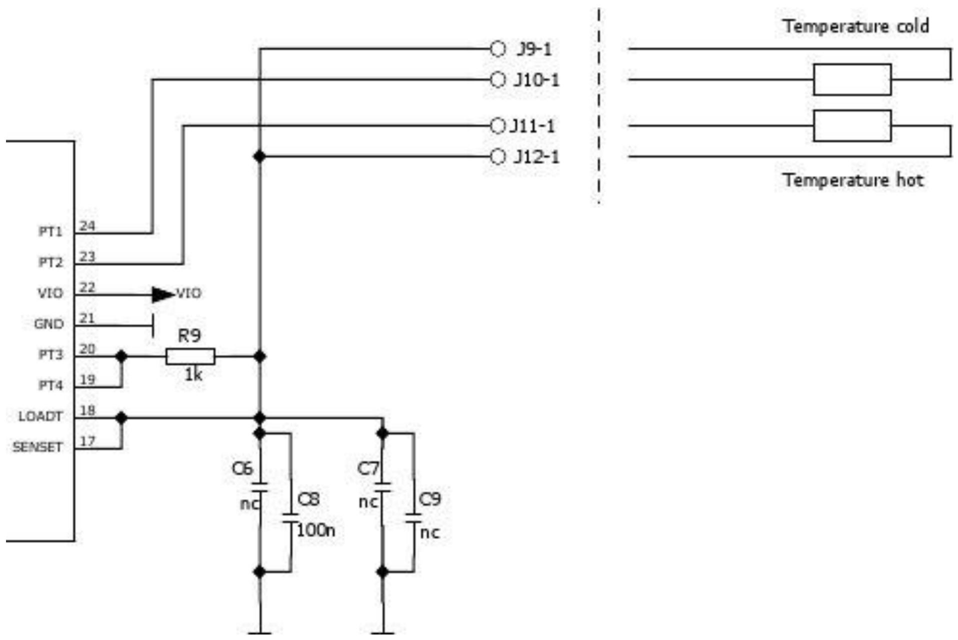
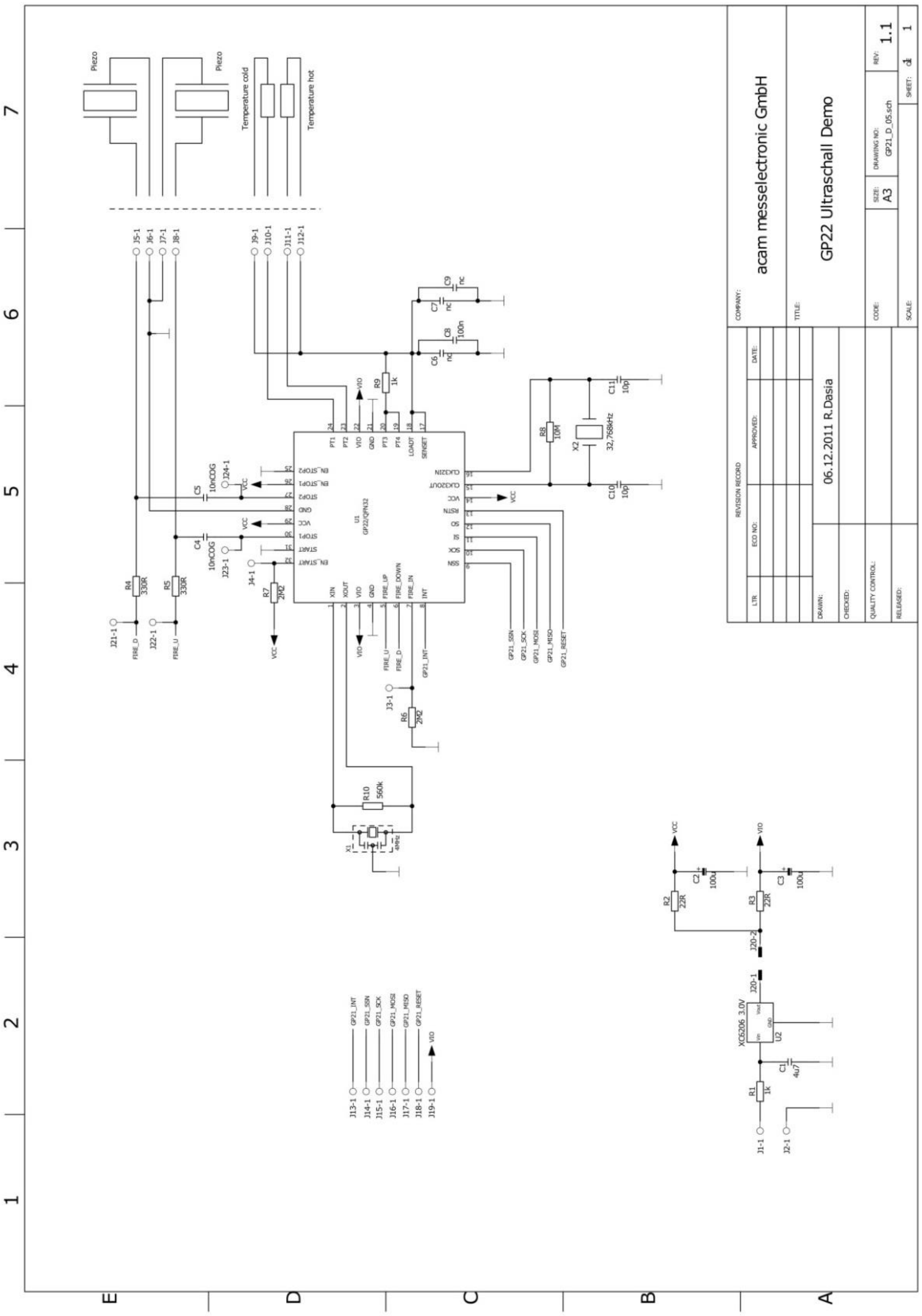


Figure 2.8 Temperature measurement circuit

4 Layout & Schematics of a demo board

4.1 Full Schematics



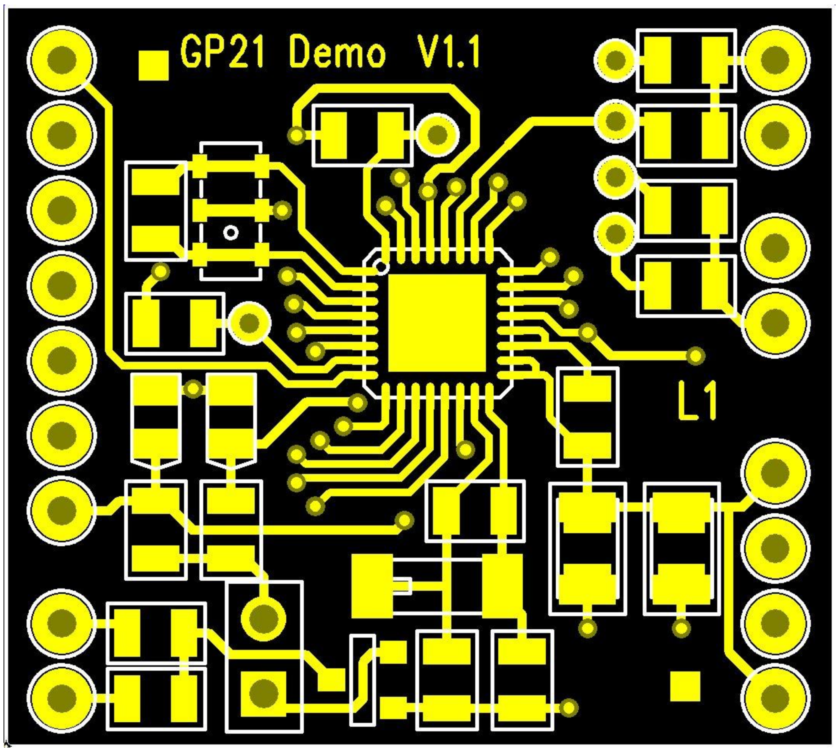
4.2 Layout

The PCB has significant influence on the measurement quality so stick as close as possible to the design and the recommendations given in this chapter. Then a 2 layer should be sufficient. The demo board is the same for TDC-GP21 and TDC-GP22, so the name on the PCB was not changed.

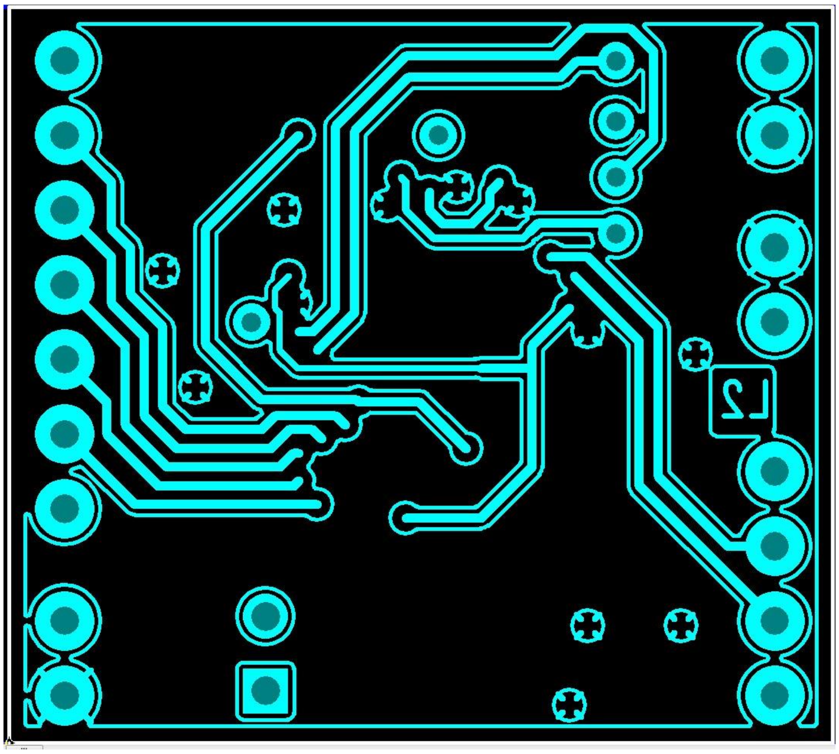
Summary of the most important layout considerations:

- Place the oscillators very close to the device
- Do NOT cross the SPI lines with the load or I/O lines
- Do NOT cross the load line with the oscillator lines
- Place the bypassing capacitors C2 and C3 close to the supply voltage pins of the GP22
- Keep the temperature port lines as short and symmetric as possible
- If possible, use flooded planes around the oscillators

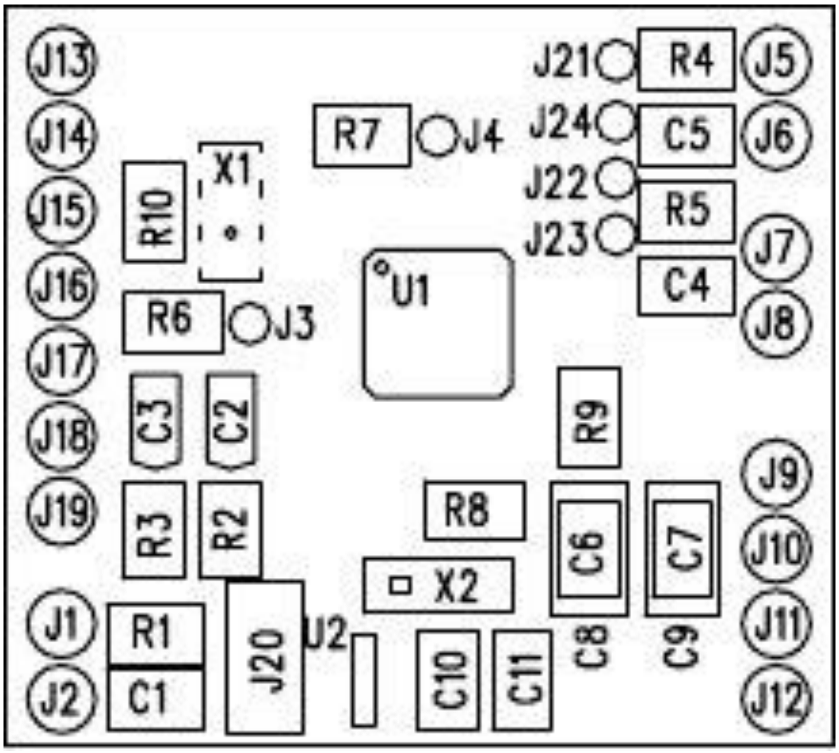
4.2.1 Top Layer



4.2.2 Bottom Layer



4.2.3 Component Placement



5 Example configuration

The following chapter gives an example configuration for a heat meter application and describes the settings of all relevant parameters. A detailed description of the configuration bits is further provided in section 3 of the TDC-GP22 datasheet.

Register	Value	Typical example configuration
Register 0	'h430BE800	ANZ_FIRE = 20 (see register 6, too) DIV_FIRE = 3, fire pulse frequency = 4 MHz/4 = 1.0 MHz ANZ_PER_CALRES = 0, the 4 MHz is calibrated by a 61.035µs measurement DIV_CLKHS = 0, the 4 MHz ceramic oscillator is internally used as it is START_CLKHS = 2, the ceramic oscillator has 480 µs to settle ANZ_PORT = 1, use all 4 ports for the temperature measurement TCYCLE = 1, 512 µs cycle time for the temperature measurement ANZ_FAKE = 1, 7 fake measurements SEL_ECLK_TMP = 1, use 4 MHz for the temperature measurement cycle definition CALIBRATE = 1, mandatory in measure mode 2 to be on NO_CAL_AUTO = 0, mandatory in measure mode 2 to have auto-calibration MESSB2 = 1, switch on measure mode 2 for measuring > 2 µs. NEG_STOP/NEGSTART = 0, all set to rising edges ID0 = h00

Register 1	'h21444000	HIT2 = 2, HIT1 = 1: calculate 1. Stop - Start in measure mode 2
		EN_FAST_Init = 0, off
		HITIN2 = 0
		HITIN1 = 4, measure 3 stops (in measure mode 2 this includes the start, too, giving 4 hits)
		CURR32K = 0, use default
		SEL_START_FIRE = 1, use the internal direct wiring from the fire pulse buffer to the TDC start
		SEL_TSTO2 = 0, EN_START active
		SEL_TSTO1 = 0, FIRE_IN pin is used as fire in
		ID1 = h00

Register 2	'hA0138800	EN_INT = b0101, interrupt given by time_out, ALU ready or Timeout. (see also register 6) RFEDGE1 = RFEDGE2 = 0, use only rising edges DELVAL1 = 5000, the first stop is accepted after 39.0625 µs ID2 = h00

Register 3	'hD0A24800	EN_AUTOCALC_MB2 = 1, automatic calculation of the sum of RES_0, RES_1 and RES_2. This calculation does not increase the address pointer. EN_FIRST_WAVE = 1, first hit detection mechanism is enabled EN_ERR_VAL = 0, there is enough time to read the status register SEL_TIMO_MB2 = 2 , time out is generated after 1024 µs DELREL1 = 8, DELREL2 = 9, DELREL3 = 10, measure the 8th, 9th and 10th stop after the first hit ID3 = h00

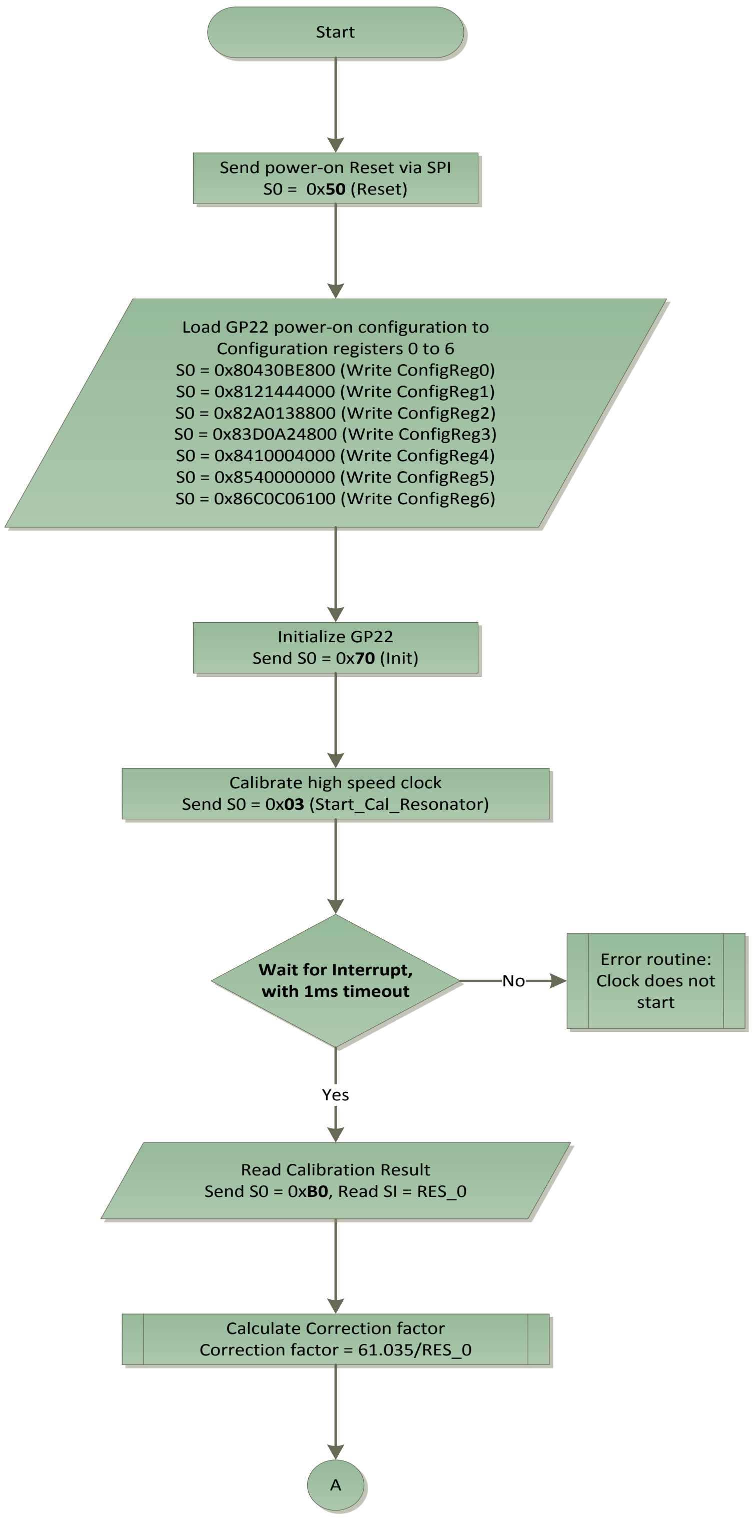
Register 4	'h10004000	DIS_PW = 0, pulse width measurement is not disabled
		EDGE_PW = 0, pulse width measured on rising edge
		OFFSRNG1 = 0, no negative offset
		OFFSRNG2 = 1, OFFS = 0: total offset = 20 mV + 0 mV = 20 mV
		ID4 = h00

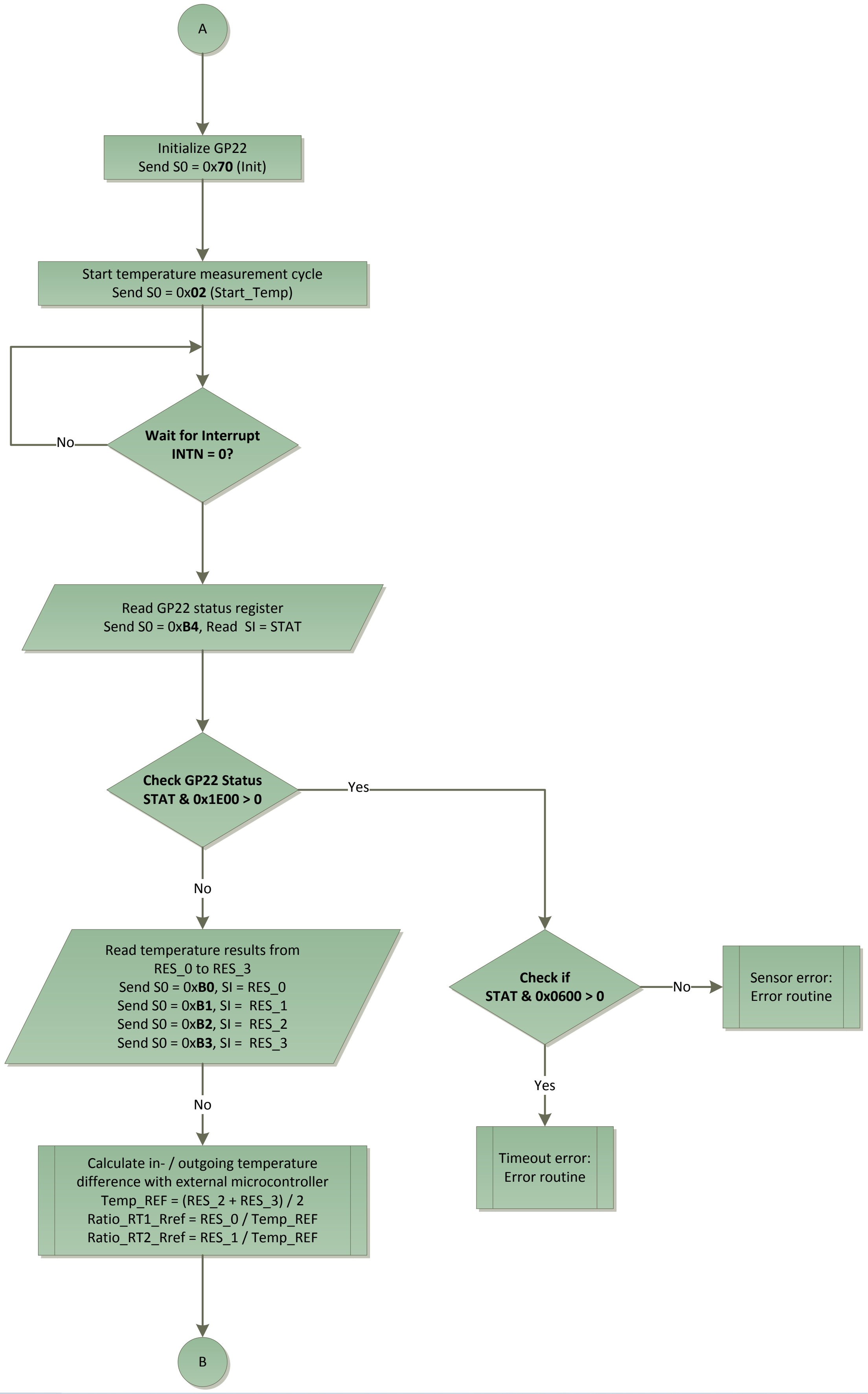
Register 5	'h40000000	CON_FIRE = 2, enable FIRE_UP. If opcode Start_TOF_Restart is used FIRE_UP and FIRE_DOWN are used alternately for up and down flow measurements. With the configuration described here the measurement cycle starts sending fire pulses at pin FIRE_UP.
		EN_STARTNOISE = 0, switch off
		DIS_PHASESHIFT = 0, phase noise unit is active to improve the statistical behavior
		REPEAT_FIRE = 0, no sing-arround
		PHASE_FIRE = 0, no phase change in the fire pulse sequence
		ID5 = h00

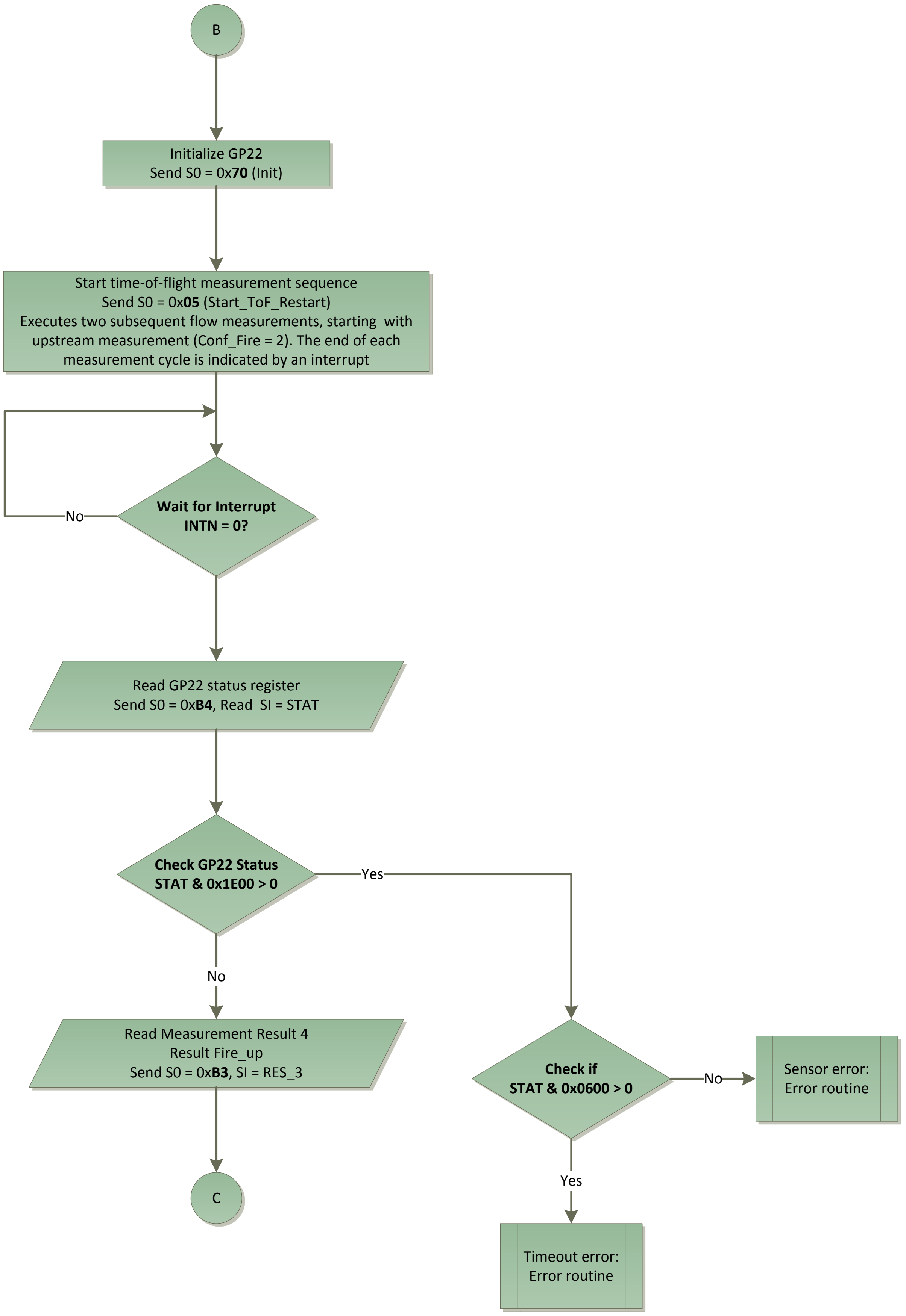
Register 6	'hC0C06100	EN_ANALOG = 1, use the internal analog circuit NEG_STOP_TEMP = 1, use the internal Schmitt trigger for the temperature measurement DA_KORR = 0, offset is set in register 4 TW2 = 3, 300 µs delay to charge up the capacitors of the highpass EN_INT = b1101, interrupt given by time_out, ALU ready or end of EEPROM action (see also register 2) START_CLKHS = 2, the ceramic oscillator has 480 µs to settle (see also register 0) CYCLE_TEMP = 0, use factor 1.0 for the Start_Temp_Restart CYCLE_TOF = 0, use factor 1.0 for the delay between two TOF measurements HZ60 = 0, 50 Hz base FIRE0_DEF = 1, mandatory when using the internal analog circuit QUAD_RES = 1, use 23 ps BIN DOUBLE_RES = 0 TEMP_PORTDIR = 0, standard order for T measurement ANZ_FIRE = 20 (see register 0, too) ID6 = h00

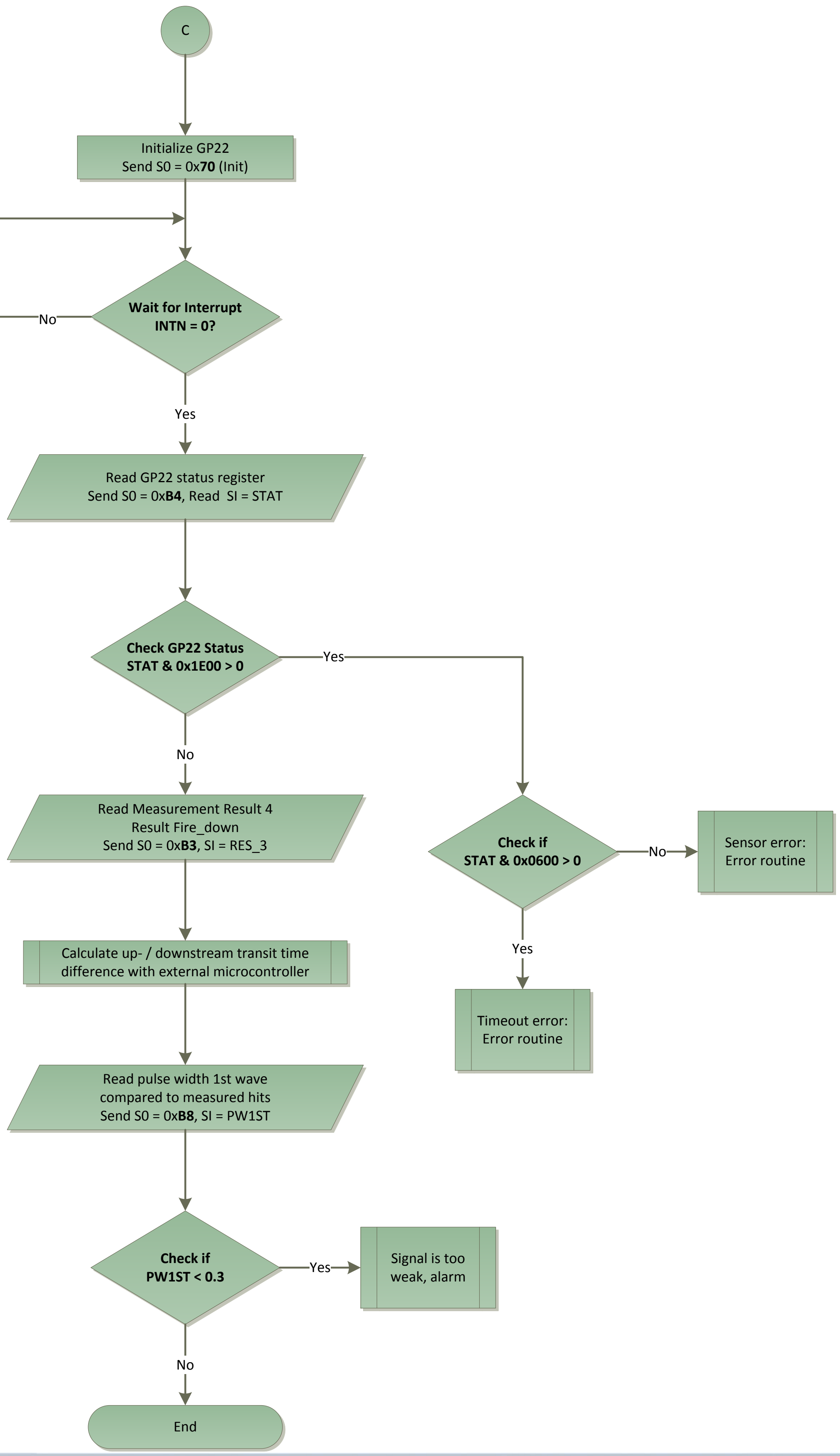
6 Software Implementation

This chapter describes a typical measurement flow. Error routines or plausibility checks of the measured results are not within the scope of this example.


Start
Send power-on Reset via SPI
S0 = 0x50 (Reset)
Load GP22 power-on configuration to
Configuration registers 0 to 6
S0 = 0x80430BE800 (Write ConfigReg0)
S0 = 0x8121444000 (Write ConfigReg1)
S0 = 0x82A0138800 (Write ConfigReg2)
S0 = 0x83D0A24800 (Write ConfigReg3)
S0 = 0x8410004000 (Write ConfigReg4)
S0 = 0x8540000000 (Write ConfigReg5)
S0 = 0x86C0C06100 (Write ConfigReg6)
Initialize GP22
Send S0 = 0x70 (Init)
Calibrate high speed clock
Send S0 = 0x03 (Start_Cal_Resonator)
Wait for Interrupt, with 1ms timeout
No
Error routine: Clock does not start
Yes
Read Calibration Result
Send S0 = 0xB0, Read SI = RES_0
Calculate Correction factor
Correction factor = 61.035/RES_0
A


A
Initialize GP22
Send S0 = 0x70 (Init)
Start temperature measurement cycle
Send S0 = 0x02 (Start_Temp)
Wait for Interrupt
INTN = 0?
No
Read GP22 status register
Send S0 = 0xB4, Read SI = STAT
Check GP22 Status
STAT & 0x1E00 > 0
Yes
No
Read temperature results from
RES_0 to RES_3
Send S0 = 0xB0, SI = RES_0
Send S0 = 0xB1, SI = RES_1
Send S0 = 0xB2, SI = RES_2
Send S0 = 0xB3, SI = RES_3
Check if
STAT & 0x0600 > 0
No
Sensor error:
Error routine
Yes
Timeout error:
Error routine
No
Calculate in- / outgoing temperature
difference with external microcontroller
Temp_REF = (RES_2 + RES_3) / 2
Ratio_RT1_Rref = RES_0 / Temp_REF
Ratio_RT2_Rref = RES_1 / Temp_REF
B


B
Initialize GP22
Send S0 = 0x**70** (Init)
Start time-of-flight measurement sequence
Send S0 = 0x**05** (Start_ToF_Restart)
Executes two subsequent flow measurements, starting with upstream measurement (Conf_Fire = 2). The end of each measurement cycle is indicated by an interrupt
Wait for Interrupt INTN = 0?
No
Read GP22 status register
Send S0 = 0x**B4**, Read SI = STAT
Check GP22 Status STAT & 0x1E00 > 0
Yes
No
Read Measurement Result 4
Result Fire_up
Send S0 = 0x**B3**, SI = RES_3
C
Check if STAT & 0x0600 > 0
No
Sensor error: Error routine
Yes
Timeout error: Error routine


C
Initialize GP22
Send S0 = 0x70 (Init)
Wait for Interrupt
INTN = 0?
No
Yes
Read GP22 status register
Send S0 = 0xB4, Read SI = STAT
Check GP22 Status
STAT & 0x1E00 > 0
Yes
No
Read Measurement Result 4
Result Fire_down
Send S0 = 0xB3, SI = RES_3
Check if
STAT & 0x0600 > 0
No
Sensor error:
Error routine
Yes
Timeout error:
Error routine
Calculate up- / downstream transit time
difference with external microcontroller
Read pulse width 1st wave
compared to measured hits
Send S0 = 0xB8, SI = PW1ST
Check if
PW1ST < 0.3
Yes
Signal is too
weak, alarm
No
End

Description:

After an initial power-on reset the basic register configuration is transferred to the TDC-GP22 and the device is initialized through calling the init-instruction.

Then the Start_CAL_Resonator instruction forces the GP22 to execute a calibration of the GP22 high speed clock.

From now on the complete program flow is interrupt driven.

The external microcontroller reads the result and calculates the ratio from the measured result RES_0 and the reference value that is given by the number of 32.768 Hz clock cycles that are used for calibration. Here, we use two periods of the 32.768 kHz clock according to the ANZ_PER_CALRES parameter in the example configuration. This equals to a reference value of 61,035 µs.

After that the init instruction resets the pointer result register and the interrupt line, reinitializes the TDC-GP22 and prepares it for the next measurement.

The temperature measurement sequence is started by sending the Start_Temp opcode. By doing that the GP22 executes the temperature measurement sequence. The end is indicated by an interrupt. The microcontroller then checks the GP22 status register for errors and reads the results from result register RES_0 to RES_3 to calculate the temperature difference. The formulas are described in section 2.2 of this paper.

Again, the init instruction prepares the GP22, but now for the time-of-flight measurement sequence.

With the Start_ToF_Restart instruction the TDC-GP22 automatically executes the complete measurement sequence for a transit time measurement in upstream and downstream direction. Here, we start with the upstream measurement (CONF_FIRE = 2), sending a burst of 20 pulses (ANZ_FIRE = 20) at pin FIRE_UP. After reception of the echo signal the TDC-GP22 starts calculating the transit times from its internally stored raw values. It automatically calculates the three time intervals between the three stops and the start and also the sum of the three. An interrupt indicates the end of the calculation.

After checking the status register the result of the upstream measurement is available in result register RES_3. They are read from the external microcontroller and stored for further calculation.

After these steps the init instruction prepares the device for the downstream measurement sequence. It starts automatically (according the repetition rate that is specified by the HZ60 and CYCLE_TOF parameters in register 6) and executes the same steps as described for the upstream measurement.

The external microprocessor reads the upstream and downstream result and calculates the amount of heat by using the formulas described in section 2.1 of this paper.

Finally the PW1ST register gives the ratio of the width of the first half wave compared to the half period of the received signal. The ratio can be used as indicator for the signal strength.

7 Example Code

This is a generic example code for a complete heat meter measurement flow including clock calibration of the high speed ceramic resonator, written for a STM32 microprocessor. The whole source code can be downloaded from our download center.

```
/*******************************************************************************
* Function Name  : main
* Description    : Main program.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void main(void)
{
  ENTR_CRT_SECTION();
  /* Setup STM32 system (clock, PLL and Flash configuration) */
  SystemInit();

  EXT_CRT_SECTION();

  // Choose your Slot (SPI1, SPI2)
  void* Bus_Type = SPI1;

  /* controlled loop */
  while (Dummy_var!=11) // To control the loop, e.g. (Dummy_var!=7)
  {
    if (Dummy_var==10) Dummy_var=0; // Infinite loop

    if(configured_true==FALSE)
    {
      configured_true = TRUE;
      SPIx_GPIOs_Init(Bus_Type);
      SPIx_Interface_Init(Bus_Type);

      gp22_send_1byte(Bus_Type, Power_On_Reset);
      Dly100us((void*)5);              // 500 us wait for GP22

      // Writing to the configuration registers (CR)
      // CR0: ANZ_FIRE=d20 DIV_FIRE=d3 ANZ_PER_CALRES=d0 ANZ_PORT=1...
      gp22_wr_config_reg(Bus_Type, 0x80, 0x430BE800);
      // CR1: ...
      gp22_wr_config_reg(Bus_Type, 0x81, 0x21444000);
      // CR2: EN_INT=b0101 RFEDGE1=RFEDGE=0 DELVAL1=d5000 ID2=0
      gp22_wr_config_reg(Bus_Type, 0x82, 0xA0138800);
      // CR3: EN_AUTOCALC=1 EN_FIRST_WAVE=1 DELREL1=d8 DELREL2=d9 DELREL3=d10
      gp22_wr_config_reg(Bus_Type, 0x83, 0xD0A24800);
      // CR4: DIS_PW=0 EDGE_PW=0 OFFSRNG1=0 OFFSRNG2=1 OFFS=0 ID4=0
      gp22_wr_config_reg(Bus_Type, 0x84, 0x10004000);
      // CR5: FIRE_UP=1
      gp22_wr_config_reg(Bus_Type, 0x85, 0x40000000);
      // CR6: ...
      gp22_wr_config_reg(Bus_Type, 0x86, 0xC0C06100);
    }

    // ..........................................................................
    // ........................Calibrate High Speed Clock.......................
    // .........................Temperature Measurment..........................
    // ......................TIME OF FLIGHT MEASUREMENT.........................
    // .....................Sum result of multihit values.......................

if((Dummy_var==0) | (Dummy_var==10))
{
    //---------------------------------------------------------------------------
    // Start Calibrate High Speed Clock Cycle
    gp22_send_1byte(Bus_Type, Init);
    gp22_send_1byte(Bus_Type, Start_Cal_Resonator);

    // Wait for INT Slot_x
    if (Bus_Type==SPI1) while (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_4)==1);
    if (Bus_Type==SPI2) while (GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_11)==1);

    //Calculate Correction factor
    //The time interval to be measured is set by ANZ_PER_CALRES
    //which defines the number of periods of the 32.768 kHz clock:
    //2 periods = 61.03515625 µs
    CLKHS_freq_corr_fact = 61.03515625/
      gp22_read_n_bytes(Bus_Type, 4, 0xB0, 0x00, 16) * CLKHS_freq;

    printf("\n Correction factor for clock = %1.4f\n", CLKHS_freq_corr_fact);

    CLKHS_freq_cal = CLKHS_freq * CLKHS_freq_corr_fact; // Calibrated Clock frequency
}

if((Dummy_var==0) | (Dummy_var==5) | (Dummy_var==10))
{
    //---------------------------------------------------------------------------
    // Start Temperature Measurement Cycle
    gp22_send_1byte(Bus_Type, Init);
    gp22_send_1byte(Bus_Type, Start_Temp);

    // Wait for INT Slot_x
    if (Bus_Type==SPI1) while (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_4)==1);
    if (Bus_Type==SPI2) while (GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_11)==1);

    // Read the Status Register
    printf ("\nStatus Register = 0x%04X", gp22_read_status_bytes(Bus_Type));

    Error_Bit = gp22_status_count_error(Bus_Type);
    if (Error_Bit > 0)
    {
      gp22_analyse_error_bit(Bus_Type);
      printf("\nTemp-Measurement are skipped\n");
    }

    else
    {
      //discharge time in µs
      Temp_PT1 = gp22_read_n_bytes(Bus_Type, 4, 0xB0, 0x00, 16) / CLKHS_freq_cal;
      Temp_PT2 = gp22_read_n_bytes(Bus_Type, 4, 0xB0, 0x01, 16) / CLKHS_freq_cal;
      Temp_PT3 = gp22_read_n_bytes(Bus_Type, 4, 0xB0, 0x02, 16) / CLKHS_freq_cal;
      Temp_PT4 = gp22_read_n_bytes(Bus_Type, 4, 0xB0, 0x03, 16) / CLKHS_freq_cal;

      Temp_REF = (Temp_PT3 + Temp_PT4) / 2;

      // Calculate Temp_cold at PT1
      Ratio_RT1_Rref    = Temp_PT1/Temp_REF * corr_fact;
      R1                = Ratio_RT1_Rref * R0_at_0C;
      Temp_cold_in_Celcius = (-R0_at_0C *Coeff_A +sqrt(
                               ((R0_at_0C *Coeff_A)*(R0_at_0C *Coeff_A))
                                 -4*R0_at_0C *Coeff_B *(R0_at_0C - R1) ) )
                                   /(2 *R0_at_0C *Coeff_B);

      // Calculate Temp_hot at PT2
      Ratio_RT2_Rref    = Temp_PT2/Temp_REF * corr_fact;
      R2                = Ratio_RT2_Rref * R0_at_0C;
      Temp_hot_in_Celcius = (-R0_at_0C *Coeff_A +sqrt(
                              ((R0_at_0C *Coeff_A)*(R0_at_0C *Coeff_A))
                                -4*R0_at_0C *Coeff_B *(R0_at_0C - R2) ) )
                                  /(2 *R0_at_0C *Coeff_B);

      // Print result
      printf("\n\tPT1= %6.3fµs / PT2= %6.3fµs \n\tPT3= %6.3fµs / PT4= %6.3fµs",
             Temp_PT1, Temp_PT2, Temp_PT3, Temp_PT4);
      printf("\n Ratio RT1/Rref= %2.5f / Ratio RT2/Rref= %2.5f",
             Ratio_RT1_Rref, Ratio_RT2_Rref);
      printf("\n Temp_cold= %3.3f°C / Temp_hot= %3.3f°C\n",
             Temp_cold_in_Celcius, Temp_hot_in_Celcius);
    }

}

    //----------------------------------------------------------------------------
    // Start Time Of Flight Measurement Cycle
    gp22_send_1byte(Bus_Type, Init);

    gp22_send_1byte(Bus_Type, Start_TOF_Restart);

    Error_Bit=0;

    // Wait for INT Slot_x, UPSTREAM
    if (Bus_Type==SPI1) while (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_4)==1);
    if (Bus_Type==SPI2) while (GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_11)==1);

    // Read the Status Register
    status_byte_UP = gp22_read_status_bytes(Bus_Type);
    Error_Bit += gp22_status_count_error(Bus_Type);

    // Result_UP, TOF in µs
    // RES_0 to RES_2 to be displayed only for evaluation purposes
    // RES_3 will be used for e.g. flow calculation
    Result_0_up = gp22_read_n_bytes(Bus_Type,4,0xB0,0x00,16)/CLKHS_freq_cal;
    Result_1_up = gp22_read_n_bytes(Bus_Type,4,0xB0,0x01,16)/CLKHS_freq_cal;
```

```
Result_2_up = gp22_read_n_bytes(Bus_Type,4,0xB0,0x02,16)/CLKHS_freq_cal;
average_Result_up = gp22_read_n_bytes(Bus_Type,4,0xB0,0x03,16)/CLKHS_freq_cal;

gp22_send_1byte(Bus_Type, Init);

// Wait for INT Slot_x, DOWNSTREAM
if (Bus_Type==SPI1) while (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_4)==1);
if (Bus_Type==SPI2) while (GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_11)==1);

// Read the Status Register
status_byte_DOWN = gp22_read_status_bytes(Bus_Type);
Error_Bit += gp22_status_count_error(Bus_Type);

// Result_DOWN, TOF in µs
// RES_0 to RES_2 to be displayed only for evaluation purposes
// RES_3 will be used for e.g. flow calculation
Result_0_down = gp22_read_n_bytes(Bus_Type,4,0xB0,0x00,16)/CLKHS_freq_cal;
Result_1_down = gp22_read_n_bytes(Bus_Type,4,0xB0,0x01,16)/CLKHS_freq_cal;
Result_2_down = gp22_read_n_bytes(Bus_Type,4,0xB0,0x02,16)/CLKHS_freq_cal;
average_Result_down = gp22_read_n_bytes(Bus_Type,4,0xB0,0x03,16)/ CLKHS_freq_cal;

if (Error_Bit > 0)
{
  gp22_analyse_error_bit(Bus_Type);
  printf("\nTOF-Measurement are skipped\n");
}

else

{
  //----------------------------------------------------------------------------
  // Result after two measurements (first UPSTREAM then DOWNSTREAM)
  // Calculate UP- / DOWNSTREAM transit time difference with MCU

  // Devider for multihit sum (1..3)
  average_Result_up /= 3;
  average_Result_down /= 3;

  // discharge time in ns
  Result_0_up *= 1000;
  Result_1_up *= 1000;
  Result_2_up *= 1000;
  average_Result_up *= 1000;

  Result_0_down *= 1000;
  Result_1_down *= 1000;
  Result_2_down *= 1000;
  average_Result_down *= 1000;

  TOF_up = average_Result_up;
  TOF_down = average_Result_down;
  PW1ST += gp22_read_n_bytes(Bus_Type, 1, 0xB0, 0x08, 7);

  // print results
  printf("\nStatus Byte Up= 0x%04X / Down= 0x%04X",
         status_byte_UP, status_byte_DOWN);
```

```
      printf("\n Result_0 \tUp= %6.3f ns / Down= %6.3f ns",
             Result_0_up, Result_0_down);
      printf("\n Result_1 \tUp= %6.3f ns / Down= %6.3f ns",
             Result_1_up, Result_1_down);
      printf("\n Result_2 \tUp= %6.3f ns / Down= %6.3f ns",
             Result_2_up, Result_2_down);
      printf("\n Result_3 \tUp= %6.3f ns / Down= %6.3f ns",TOF_up, TOF_down);

      Time_of_flight_diff = TOF_up - TOF_down;
      printf("\n\tDifference RES_3(Up-Down)= %6.3f ns", Time_of_flight_diff);

      Time_of_flight_sum = TOF_up + TOF_down;
      printf("\n\tSum        RES_3(Up+Down)= %6.3f ns\n", Time_of_flight_sum);

      // to add up
      TOF_diff_sum += Time_of_flight_diff;
      TOF_diff_square_sum = TOF_diff_square_sum
                          +(Time_of_flight_diff * Time_of_flight_diff);
      TOF_sum_sum += Time_of_flight_sum;
      sum_counter ++;

      if (sum_counter > no_of_avg)    // Output after no_of_avg measurements
      {
        TOF_diff_avg = TOF_diff_sum / no_of_avg;
        printf("\n  Avg. value of difference = %6.3f ns", TOF_diff_avg);
        Std_Dev_of_Diff = sqrt( (TOF_diff_square_sum
                          -(TOF_diff_square_sum/no_of_avg)) / (no_of_avg-1));
        printf("\n  Std.Dev. of Diff. = %6.1f ps", Std_Dev_of_Diff*1000);
        printf("\n  PW1ST = %1.2f\n", PW1ST/(no_of_avg));

        Std_Dev_of_Diff = 0;
        TOF_diff_sum = 0;
        TOF_diff_square_sum = 0;
        sum_counter = 1;
        PW1ST = 0;
      }
    }

      Dummy_var++; // To Control the loop
  } // End while Dummy_var

} //End main
```

8 Revision History

04.12.2012	First Edition
28.01.2013	Version 1.1 for release, section 2.2 Gain Correction Factor added, section 5 Register 5 Fire Up/Down corrected, section 7 Source Code with Gain Correction Factor modified
10.07.2013	Version 1.2 for release, section 6 Flowchart Up/Down measure direction corrected, section 7 Example Code including part "Calibrate High Speed Clock" modified and the print-function moved to the end of the ToF Measurement

mess•electronic

acam-messelectronic gmbh
Friedrich-List-Straße 4
76297 Stutensee-Blankenloch
Germany
Phone +49 7244 7419 – 0
Fax +49 7244 7419 – 29
E-Mail support@acam.de
www.acam.de