Application Note

AN000565

TDC-GP30 Volume 3 User Manual

Details of Operation

v1-00 • 2018-Nov-16

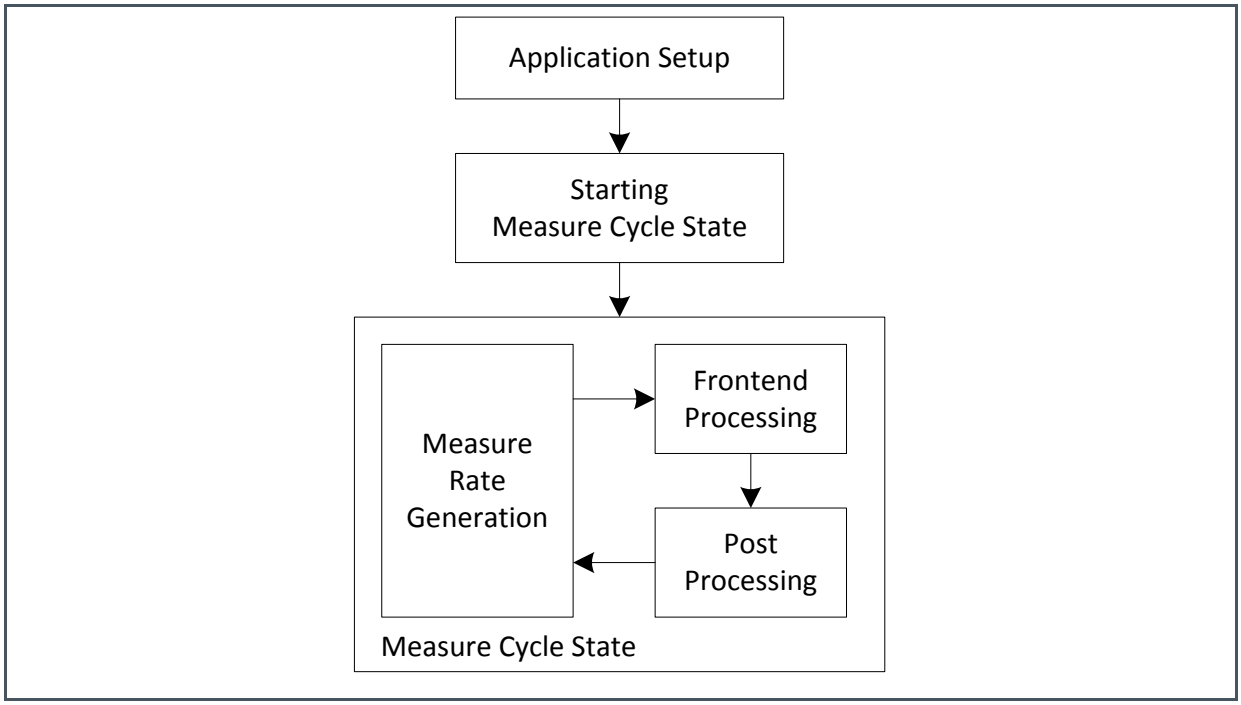
Content Guide

1 Introduction

1.1 Ultrasonic Flow Converter (UFC) Processes

When integrating **ams** UFCs (TDC-GP21 / -GP22 / -GP30) into water, gas or heat meter applications there is always a basic set of processes needed for the ultrasonic and temperature measurements:

Figure 1:
Basic Processes



- Application Setup — Initial configuration with application parameters
- Measure Rate Generation — Controlling the ultrasonic and temperature sensors
- Frontend Processing — TDC measurement of ultrasonic and temperature sensors
- Post Processing — Conversion of time based frontend results into flow and temperature results

In applications realized with **ams** TDC-GP21/-GP22 the “Frontend Processing” is the only process executed by those chips themselves and all other processes have to be executed by a remote controller.

Compared to TDC-GP21/-GP22, the TDC-GP30 has two new integrated units which allow the execution of all processes by the device itself:

- Measure Rate Generator
- CPU with programmable Firmware
  - Performing “Application Setup”
  - Performing “Post Processing”

1.2 Operating Modes

The TDC-GP30 is able to operate in following process combinations:

Figure 2:
Operating Modes

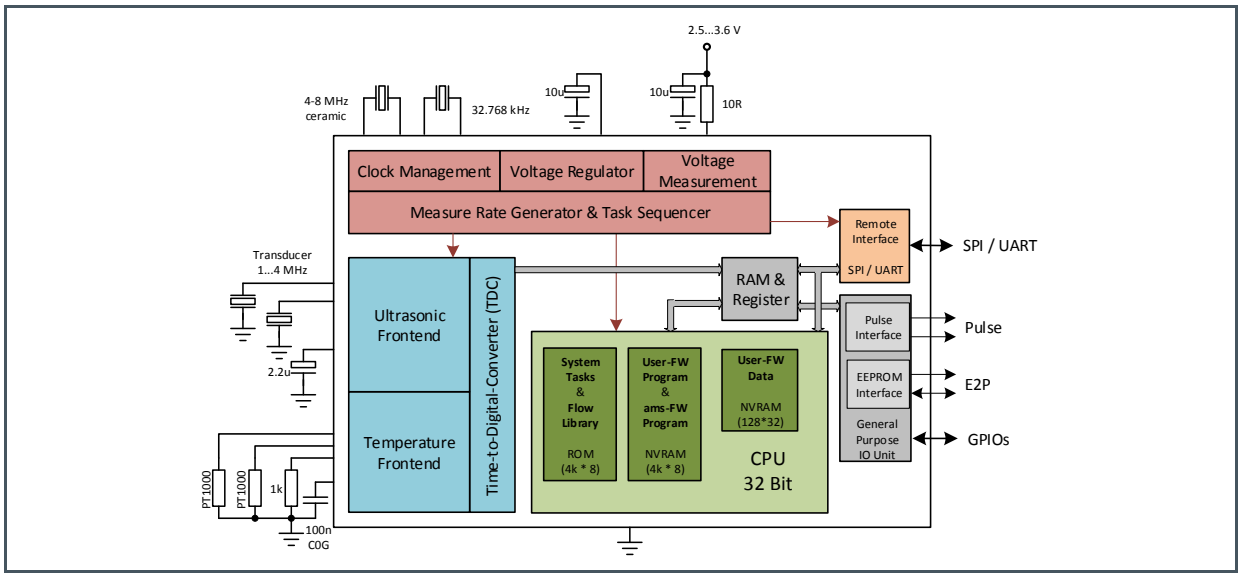
Application Setup	Measure Rate Generation	Post Processing	Operating Modes
By TDC-GP30	By TDC-GP30	By TDC-GP30	Flow meter mode
Per remote		Per remote	Time conversion mode (self-controlled)
	Per remote		Time conversion mode (remote-controlled)
All other combinations			Only for test or debug purpose

Frontend processing is always performed by TDC-GP30.

1.2.1 Flow Meter Mode

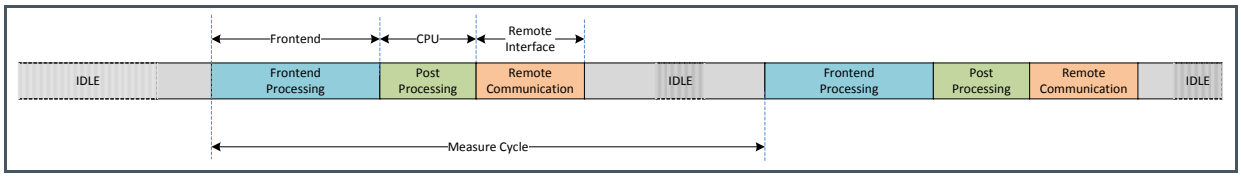
Flow meter mode is the typical application mode for TDC-GP30, where all processes are performed by TDC-GP30. The application setup is performed by a bootloader, executed in the ROM of the integrated CPU. The post processing is realized by a programmed firmware, also executed in the integrated CPU.

Figure 3:
Block Diagram



The coordination of the tasks together with the remote communication in measure cycle state is performed by the “Measure Rate Generator & Task Sequencer”.

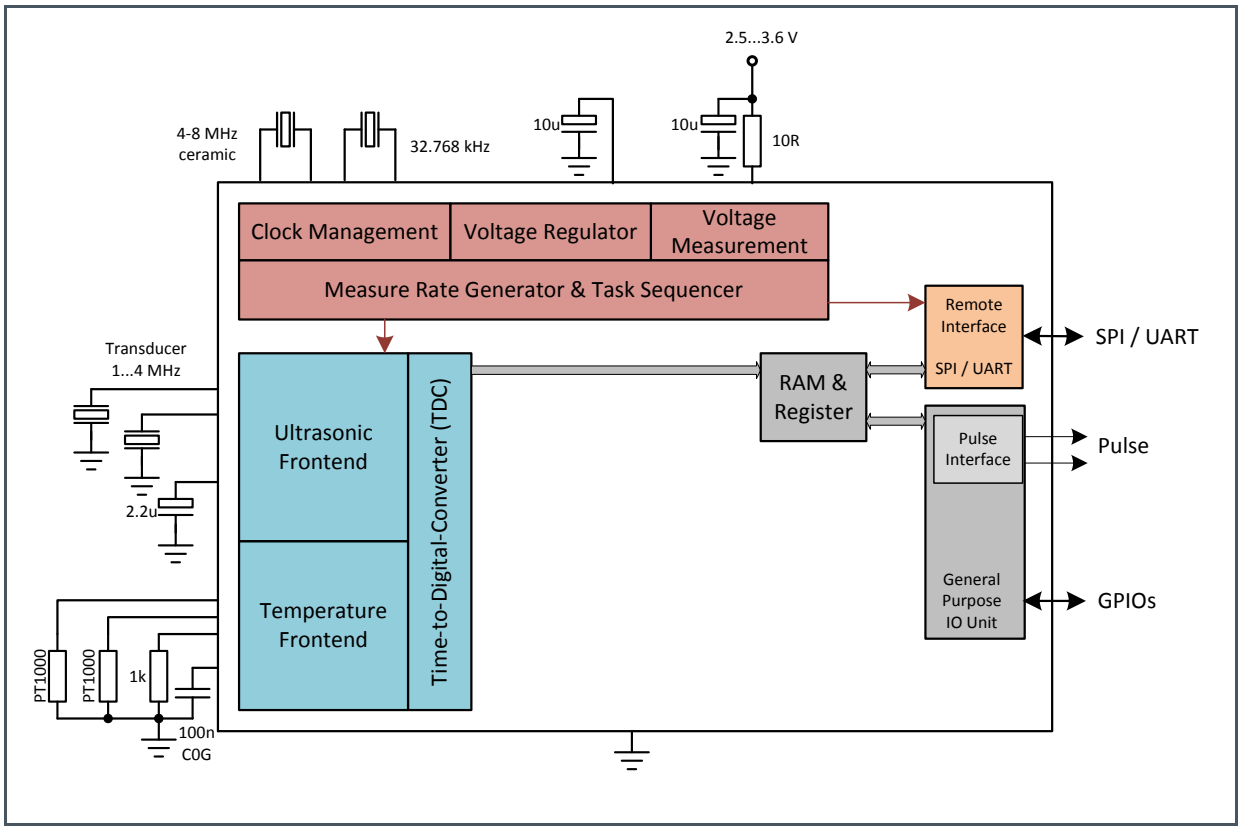
Figure 4:
Processes in Flow Meter Mode



1.2.2 Time Conversion Mode

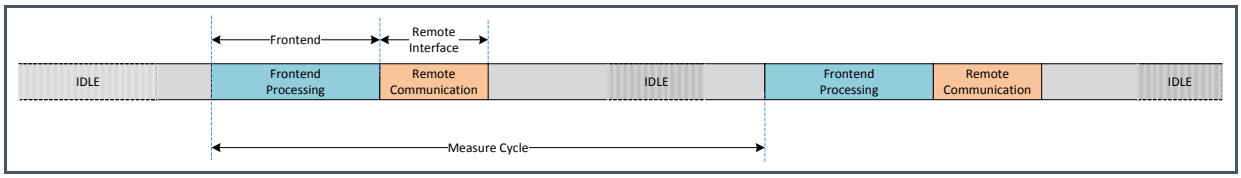
In time conversion mode the application setup and post processing are performed by a remote controller. In case the remote controller also performs the measure rate generation, the GP30 operates in the same mode as GP21/GP22.

Figure 5:
Active Blocks in Time Conversion Mode



In time conversion mode, the post processing in integrated CPU is not involved in GP30 task sequence. Also the I²C interface cannot be used, because it is only controllable by programmed firmware in CPU.

Figure 6:
GP30 Processes in Time Conversion Mode



A post processing can be performed by the remote controller after the „Remote Communication“ has been finished.

1.3 Application Setup

The application setup of TDC-GP30 hardware is defined by two sections in the register area:

- Configuration registers (**CR**) (0x0C0 – 0x0CE)
- System handling registers (**SHR**) (0x0D0 – 0x0DD)

Both register sections will be reset after the execution of a “System Reset”.

A detailed description of these register sections can be found in the GP30 data sheet volume 1.

The following sections show examples of how to configure GP30 for different applications.

1.3.1 Configuration Registers

The various parameters in the configuration register have the character of constants. They are typically defined once before the measure cycle state is started and are typically not changed while GP30 is in measure cycle state.

Nevertheless, it is also possible to update parameters in the configuration registers by a remote controller or the integrated CPU during measure cycle state, e.g. to change the ToF measurement rate.

Application setup for:

- **Time Conversion Mode**
  The configuration data has to be downloaded into this section via remote interface before starting the measure cycle state.
- **Flow Meter Mode**
  The configuration data has to be stored in the the “Configuration Data” memory block in NVRAM for FW Data (0x16C – 0x17A).
  In case firmware is released by writing the release code to address 0x17B, the configuration data will be transferred by the bootloader from this memory block to the configuration registers after each “System Reset”.

1.3.2 System Handling Registers

The parameters in the system handling register have a dynamic character. They are typically updated by post processing during measure cycle state, but have to be initially configured before the measurement starts.

Whether all registers of this section need to be initialized or not depends on the customer application. More details are given in the following chapters.

Application setup for:

- **Time Conversion Mode**
  The initial system handling data has to be downloaded to this section together with the data for configuration register via remote interface before the measure cycle state is started
- **Flow Meter Mode**
  Initialization of system handling registers is usually performed by the firmware. **ams** firmware uses the subroutine “FW_INIT”, executed once after bootloader has finished. The FW_INIT subroutine also handles initialization of FW parameters which are only allocated to FW program code.

1.4 Starting Measure Cycle State

The measure cycle state is started in the following ways:

- **Time Conversion Mode**
  The measure cycle state can be started by the remote command **RC_MCT_ON**, “Measure Cycle Timer On” coded with 0x8B.
- **Flow Meter Mode**
  The measure cycle state is started by bootloader if the task sequencer cycle time **MR_CT** (in configuration register **CR_MRG_TS**) is configured to a value greater than zero.

In both modes, the measure cycle state can be stopped by remote command **RC_MCT_OFF**, “Measure Cycle Timer Off”. It is restarted by **RC_MCT_ON**, “Measure Cycle Timer On”.

After a power-on, the low speed clock needs enough time for upcoming. Therefore, the measure cycle state is released earliest after 2 seconds.

1.5 Measure Rate Generation & Task Sequencer

Block diagram Figure 8 shows the interaction between the Measure Rate Generator and the Task Sequencer.

The Measure Rate Generator supplies up to 7 different measure task requests, which can trigger the task sequencer. The Measure Rate Cycle Timer is the central function in the Measure Rate Generator and generates two measure cycle triggers with a phase difference of a half cycle, see Figure 7.

Figure 7:
Cycle Triggers

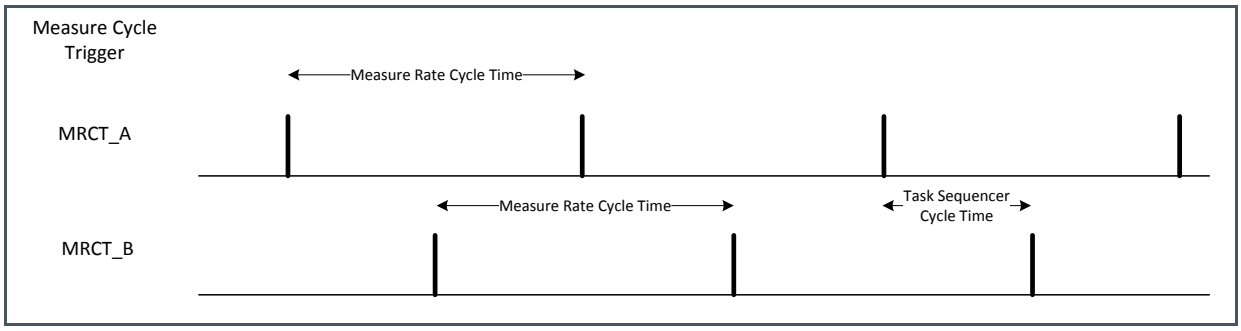
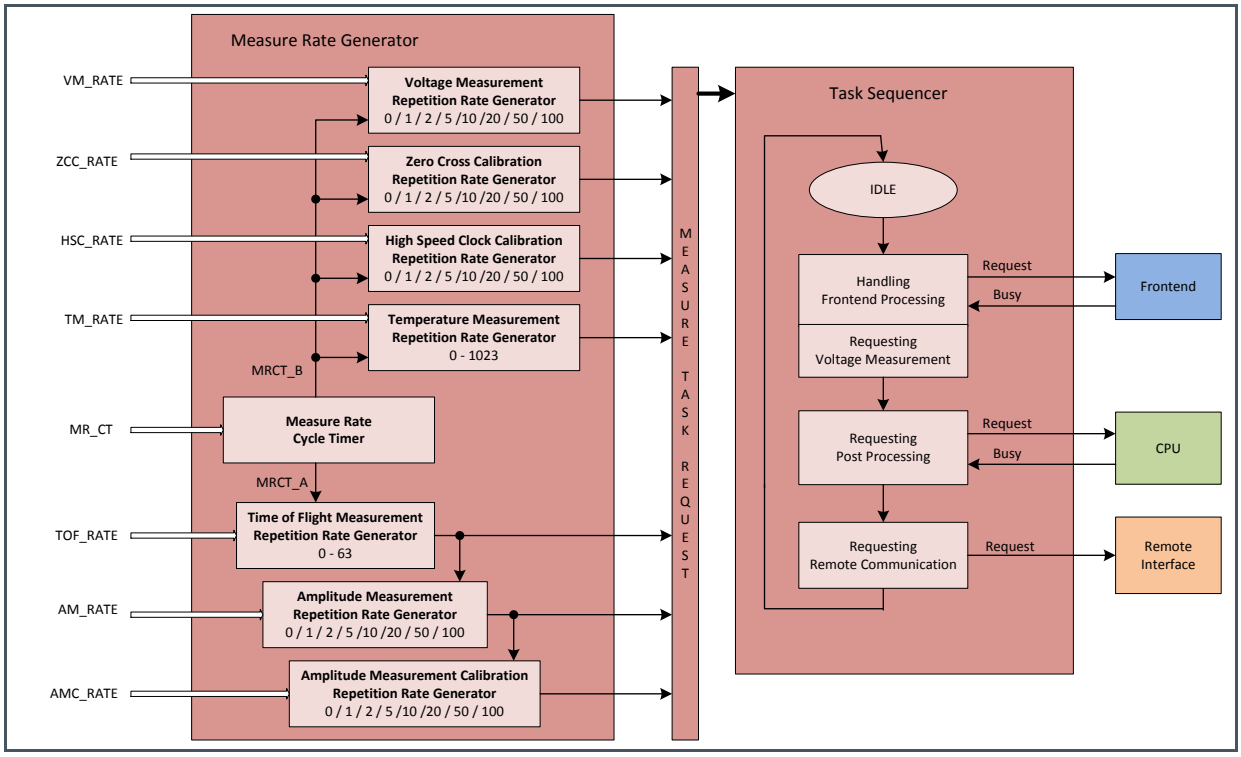


Figure 8:
Measure Rate Generation & Task Sequencer



The measure rate cycle time is configured by **MR_CT** in register **CR_MRG_TS**. Since the measure rate cycle is divided into two parts A and B, starting at **MRCT_A** and **MRCT_B**, respectively, the task sequencer can use half the measure rate cycle time to perform a complete task sequence cycle.

Each measure task has its own repetition rate generator and is assigned to one of the two measure cycle triggers **MRCT_A** or **MRCT_B**, whereby **MRCT_A** triggers the ultrasonic measurement tasks and MRCT_B triggers all other measurement tasks.

Figure 9:
Important Registers

Register	Address	Parameters
CR_MRG_TS	0x0C6	**MR_CT**: Defines measure rate cycle time
CR_CPM	0x0C5	**HSC_RATE**: Defines repetition rate for high speed clock calibration on **MRCT_B** **VM_RATE**: Defines repetition rate for voltage measurement on **MRCT_B**
CR_TM	0x0C7	**TM_RATE**: Defines repetition rate for temperature measurement on **MRCT_B**
CR_USM_FRC	0x0C9	**ZCC_RATE**: Defines repetition rate for zero cross calibration on **MRCT_B**
CR_USM_AM	0x0CB	**AM_RATE**: Defines repetition rate for amplitude measurement on **MRCT_A** **AMC_RATE**: Defines repetition rate for amplitude measurement calibration on **MRCT_A**
SHR_TOF_RATE	0x0D0	**TOF_RATE**: Defines repetition rate for time of flight measurement on **MRCT_A**

The cycle time of the different measurement tasks can be determined as follows:

- Cycle Time (TOF) = **MR_CT** * **TOF_RATE**
- Cycle Time (AM) = **MR_CT** * **TOF_RATE** * **AM_RATE**
- Cycle Time (AMC) = **MR_CT** * **TOF_RATE** * **AM_RATE** * **AMC_RATE**
- Cycle Time (TM) = **MR_CT** * **TM_RATE**
- Cycle Time (HSC) = **MR_CT** * **HSC_RATE**
- Cycle Time (ZCC) = **MR_CT** * **ZCC_RATE**
- Cycle Time (VM) = **MR_CT** * **VM_RATE**

Figure 10:
Typical Application

Parameter	Value	Comment
MR_CT	256	Measure rate cycle time: 250 ms
TOF_RATE	1	Time of flight rate: 4 Hz
AM_RATE	1	Amplitude measurement performed with every time of flight measurement
AMC_RATE	20	Calibration of amplitude measurement every 5 s
TM_RATE	120	Temperature measurement every 30 s
HSC_RATE	50	High speed clock calibration every 12.5 s
ZCC_RATE	20	Zero cross calibration every 5 s
VM_RATE	100	Voltage measurement every 25 s

1.6 Calculating Task Sequencer Cycle Time

The following example shows how task sequencer cycles are allocated by the different measure and communication tasks.

The following calculations only define times for typical configurations and can differ for other configurations.

Figure 11:
MRCT Sequence A and B

<table>
<tr><th>MRCT</th><th colspan="2">Task</th><th></th><th>Time [ms]</th></tr>
<tr><td rowspan="6">A</td><td colspan="2">Refreshing Voltage Regulator</td><td>Always performed at start of cycle</td><td>1.7</td></tr>
<tr><td>Frontend Processing</td><td>Max. time for ultrasonic measurement with a pause time of 20 ms</td><td>For a differentiated calculation please refer to corresponding chapter</td><td>21</td></tr>
<tr><td colspan="2">Post Processing</td><td>Max. time for a firmware execution with 4000 cycles and recommended CPU speed</td><td>0.5</td></tr>
<tr><td colspan="2" rowspan="2">Remote Communication</td><td>Max. time for an SPI communication of 100 bytes payload at 10 MHz</td><td>0.1</td></tr>
<tr><td>Max. time for an UART communication of 100 bytes payload at 115200 baud</td><td>10</td></tr>
<tr><td colspan="2">IDLE</td><td></td><td></td></tr>
</table>

MRCT	Task			Time [ms]
B	Refreshing Voltage Regulator		Always performed at start of cycle	1.7
	Frontend Processing	Temperature Measurement	Typ. time for a 2-wire measurement with a pause time of 30 ms	35
		Zero Cross Calibration	Max. time if zero cross calibration is performed in this cycle	0.5
		High Speed Clock Calibration	Max. time if high speed clock calibration is performed in this cycle	0.5
	Voltage Measurement		Max. time if high speed clock calibration is performed in this cycle	2
	Post Processing		Max. time for a firmware execution with 4000 cycles and recommended CPU speed	0.5
	Remote Communication		Max. time for an SPI communication of 100 bytes payload at 10 MHz	0.1
			Max. time for an UART communication of 100 bytes payload at 115200 baud	10
	IDLE			

1.7 Clock Management

Typically the GP30 operates in low power mode. This means the TDC-GP30 is sourced all the time by a low speed oscillator (LSO) of 32.768 kHz.

The high speed oscillator (HSO) of 4 MHz, sourced by a ceramic resonator, is used for the frontend processing and is activated only when needed. Compared to a quartz, a ceramic resonator has the benefit of a short settling time which saves power consumption of TDC-GP30. On the other hand the clock needs to be calibrated periodically.

The TDC-GP30 can also be sourced with a high speed clock of 8 MHz to support ultrasonic transducers with a frequency of up to 4MHz. Bit **HS_CLK_SEL** in the configuration register **CR_CPM** serves as a flag for the bootloader in flow meter mode**.** As the default value is 1, it is necessary to use **HS_CLK_SEL** in the register **SHR_RC** for the initial communication or operation in time conversion mode.

Figure 12:
Important Registers

Register	Address	Parameters
CR_CPM	0x0C5	**HS_CLK_ST**: Defines settling time for high speed clock **HS_CLK_SEL**: 0: selects a 4 MHz clock 1: select a 8 MHz clock = default ! **HSC_RATE**: Defines repetition rate for high speed clock calibration task
SHR_RC	0x0DE	**HS_CLK_SEL**: High Speed Clock Select 00: No Change of **HS_CLK_SEL** state (WO) 01: If 4 MHz clock source, has to be initially configured after reset 10: If 8 MHz clock source 11: No Change of **HS_CLK_SEL** state (WO)

Attention

The divider is initially set after a reset, e.g. due to a watchdog trigger. In time conversion mode it is therefore necessary that the user actively takers care of the right setting of HS_CLK_SEL in SHR_RC.

1.8 Voltage Measurement

The voltage measurement is the only measurement task which is performed directly by the supervisor and not in frontend processing. It is automatically executed if **VM_RATE** > 0. The value of VDD_IO is measured and can be compared to a low battery threshold.

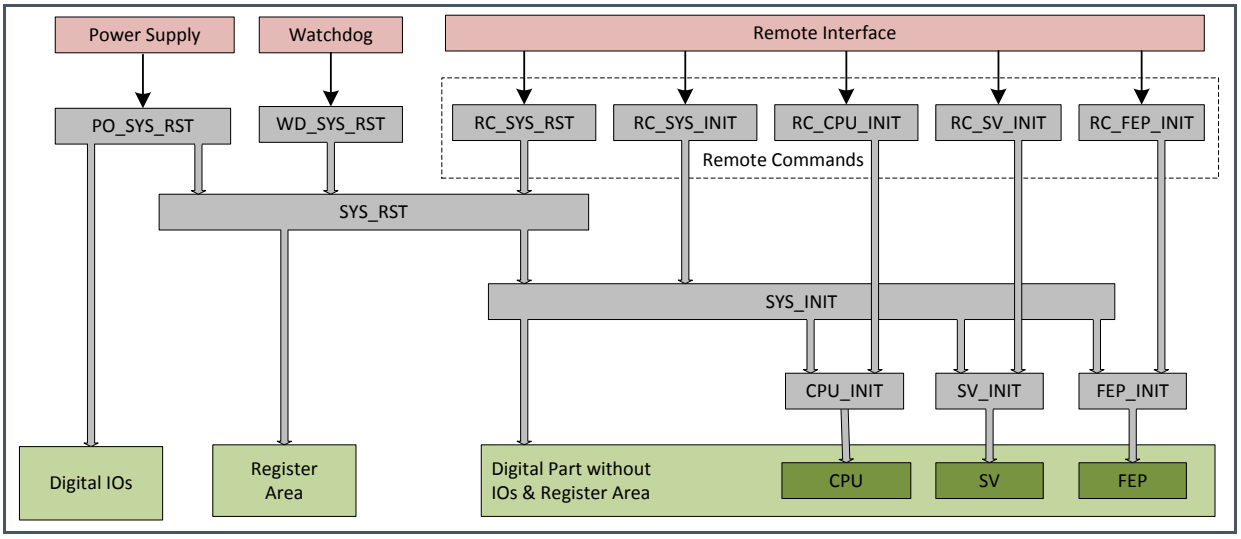
Figure 13:
Important Registers

Register	Address	Parameters
CR_CPM	0x0C5	**VM_RATE**: Defines repetition rate for voltage measurement task **LBD_TH**: Defines the low battery threshold
SRR_VDD_VAL	0x0E5	Value of VDD_IO can be read out from here

1.9 Reset Hierarchy

Resets in TDC-GP30 are initiated by turning on the power supply, by the watchdog or via remote interface commands.

Figure 14:
Reset Hierachy



Following resets can be distinguished:

- **PO_SYS_RST**
  Power-On Reset of TDC-GP30, only performed after VDD33 is switched on. Resets complete TDC-GP30 including digital IOs. After a PO_SYS_RESET, the measurement cycle timer is disabled for typically 2s as settling time for the LSO.
- **WD_SYS_RST**
  Watchdog System Reset, performed after the watchdog timer expired
  Resets complete digital part of TDC-GP30 without digital IOs
- **RC_SYS_RST**
  Remote Command System Reset, performed after sending a dedicated remote command
  Resets complete digital part of TDC-GP30 without digital IOs
- **RC_SYS_INIT**
  Remote Command System Init, performed after sending a dedicated remote command
  Resets complete digital part of TDC-GP30 without digital IOs and without register area.
  If the bootloader release code is set, system init will trigger the bootloader and thus activate the measure cycle timer as configured. This happens despite the SV_INIT, which is triggered by the SYS_INIT before the bootloader.
- **RC_CPU_INIT**
  Remote Command CPU Init, performed after sending a dedicated remote command
  Resets only CPU block of TDC-GP30

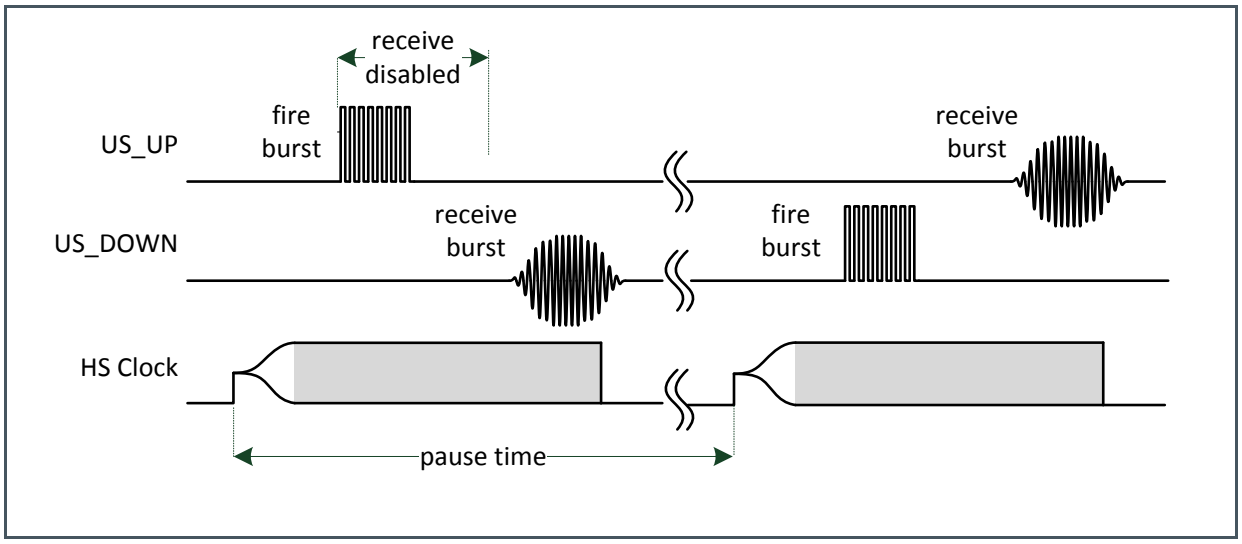
- **RC_SV_INIT**
  Remote Command SV Init, performed after sending a dedicated remote command
  Resets only supervisor (SV) block of TDC-GP30. Note that this init sets the measure cycle timer to off.
- **RC_FEP_INIT**
  Remote Command FEP Init, performed after sending a dedicated remote command
  Resets only FEP block of TDC-GP30

2 Ultrasonic Flow Measurement

2.1 Time-of-Flight Measurement

Ultrasonic flow measurement bases on two time-of-flight measurements, one in up (upstream, against flow) and one in down direction (downstream, with flow). Between both measurements different pause times can be configured in multiples of 50 Hz or 60 Hz.

Figure 15:
Send & Receive



A time-of-flight measurement (TOF) starts by first enabling the high speed oscillator (HSO). After the configured settling time of the HSO has elapsed, a configurable fire burst is sent.

To suppress noise on the receive line, typically caused by the fire burst, a disable time for the receive wave detection can be configured. Finally, as soon as all expected hits of the receive wave are detected, the HSO is disabled and the time-of-flight measurement is completed.

From a complete time-of-flight measurement (up & down) to the next one the start direction can be toggled. This helps to suppress errors by temperature drift.

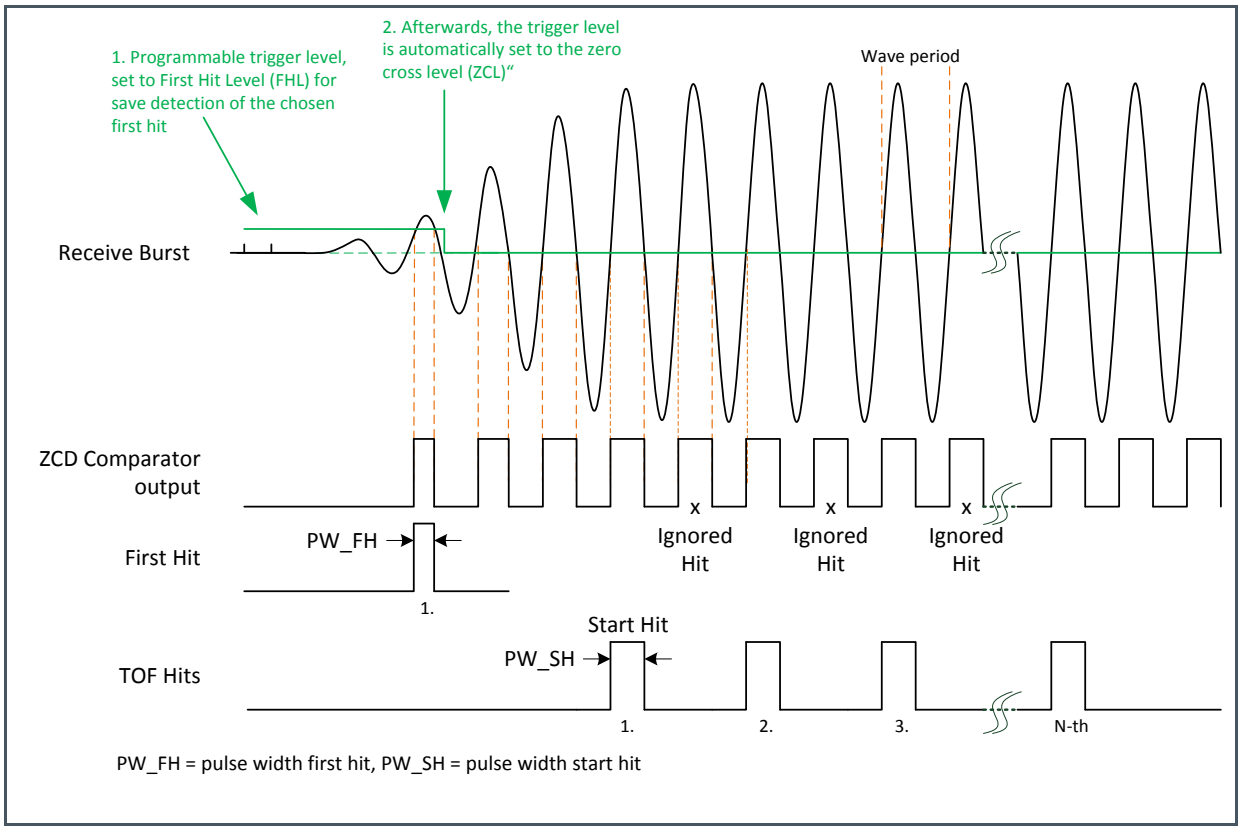
Figure 16:
Important Registers

Register	Address	Parameter
CR_CPM	0x0C5	**HS_CLK_ST**: Define settling time for high speed clock **BF_SEL**: Defines 50/60 Hz for pause time
CR_USM_PRC	0x0C8	**USM_PAUSE**: Defines pause time between two TOF measurements **USM_DIR_MODE**: Defines start direction or toggling of start direction **USM_RCV_DIS**: Defines time from start of fire burst as long receive wave detection is disabled **USM_TO**: Defines timeout condition for TOF measurement
CR_USM_FRC	0x0C9	**FPG_CLK_DIV**: Defines fire burst frequency dependent from HS clock **FPG_FP_NO** Defines number of fire pulses

2.1.1 First Wave Detection

The first wave detection is similar to the one of TDC-GP22 but with some improvements. The time-of-flight measurement is related to the first wave of the receive burst and so it is guaranteed that always the same zero crossings are used for the measurement, independent from temperature and flow.

Figure 17:
First Wave Detection



The measurement of the receive burst starts with the first wave level. The first wave level is a comparator offset other than zero cross level, e.g. 100 mV above. This offset helps to suppress noise and allows the detection of a dedicated “First Hit” that can be used as reference. Once this “First Hit” is detected the offset is set back to zero cross level.

Like in TDC-GP22, the first wave detection is complemented by a pulse width measurement option. Therefore the pulse width of the “First Hit”, measured at first wave level, is compared to the pulse width of the TOF “Start Hit” measured at zero cross level. The result is read as a pulse width ratio **PW_FH/PW_SH**. The ratio can be used to track the first wave levels for up and for down direction. It is proposed to use wide-bandwidth transducers with low quality factors, which permit a pulse with ratio **PW_FH/PW_SH** of 0.6 to 0.7.

Figure 18:
Important Registers

Register	Address	Parameter
FDB_US_PWR_U	0x081	Pulse width ratio for up direction can be read out from here
FDB_US_PWR_D	0x085	Pulse width ratio for down direction can be read out from here
CR_USM_FRC	0x0C9	**ZCD_FWL_POL**: Defines first wave level polarity (above/below zero cross level)
CR_USM_AM	0x0CB	**PWD_EN**: Enables pulse width detection
SHR_ZCD_FWL_U	0x0DA	Defines first wave level offset for up direction
SHR_ZCD_FWL_D	0x0DB	Defines first wave level offset for down direction

2.1.2 TOF Measurement

For the time-of-flight measurement itself it is necessary to define the start hit, the number of hits, and whether hits are ignored within the receive burst. The measurement themselves are done at zero crossing.

Two methods are implemented in TDC-GP30 to define the TOF “Start Hit”:

- “Start Hit” is defined by a number of hits counted from “First Hit”
- “Start Hit” is defined by a delay related to first rising edge of fire burst

The second method might be an option in case of narrow-bandwidth transducers. Such transducers will need many waves to reach the full amplitude, and the difference in amplitude from wave to wave is so small that the first wave detection cannot be used. In general, wide-bandwidth transducers with steep receive burst envelopes and consequently fast amplitude growth are preferred.

Figure 19:
Important Registers

Register	Address	Parameter
CR_USM_TOF	0x0CA	**TOF_START_HIT_MODE**: Selects “Start Hit” mode **TOF_START_HIT_NO**: Defines number of hits after “First HIT” which is dedicated as “Start Hit” **TOF_HIT_NO**: Defines number of TOF hits used for TDC measurement **TOF_HIT_IGN**: Defines number of hits ignored between two TOF hits
SHR_START_HIT_DLY	0x0D8	Defines delay value for “First Hit” related to fire burst

2.1.3 Zero Cross Offset Calibration

The zero cross level, indicated in Figure 10 above, is defined in the system handling register **SHR_ZCD_LVL**. It is automatically calibrated to the chip’s reference level voltage Vref if zero cross calibration is enabled to any rate (by setting **ZCC_RATE** > 0).

As an additional option, the zero cross level can be set manually via **SHR_ZCD_LVL**, if zero cross calibration is disabled (**ZCC_RATE** = 0).

Figure 20:
Important Registers

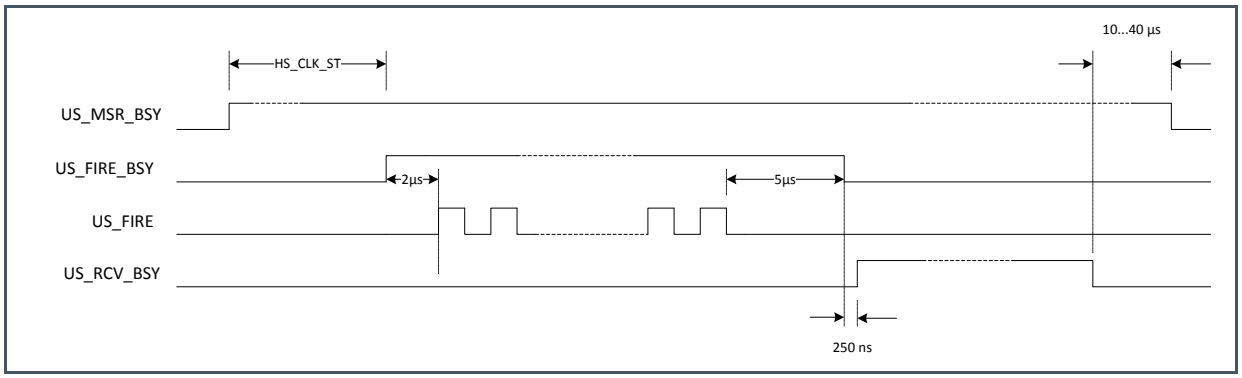
Register	Address	Parameter
CR_USM_FRC	0x0C9	**ZCC_RATE**: Defines repetition rate for zero cross calibration
SHR_ZCD_LVL	0x0D9	Defines zero cross detection level

2.1.4 Supporting Gas Meter Applications

For the support of gas meter applications some internal ultrasonic signals can be directed to GPIOs:

- GPIO0 US_FIRE Fire Burst
- GPIO1 US_DIR Direction
  0: Fire UP
  1: Fire DOWN
- GPIO4 US_FIRE_BUSY Fire Busy
- GPIO6 US_RCV_BUSY Receive Busy

Figure 21:
FEP Signals which Are Available over GPIOs to Support Gas Meter Applications. Timings Shown Are Based on HSO = 4 MHz



End of US_RCV_BUSY is reached when the last hit is measured. Therefore this signal depends on time of flight and number of measured hits.

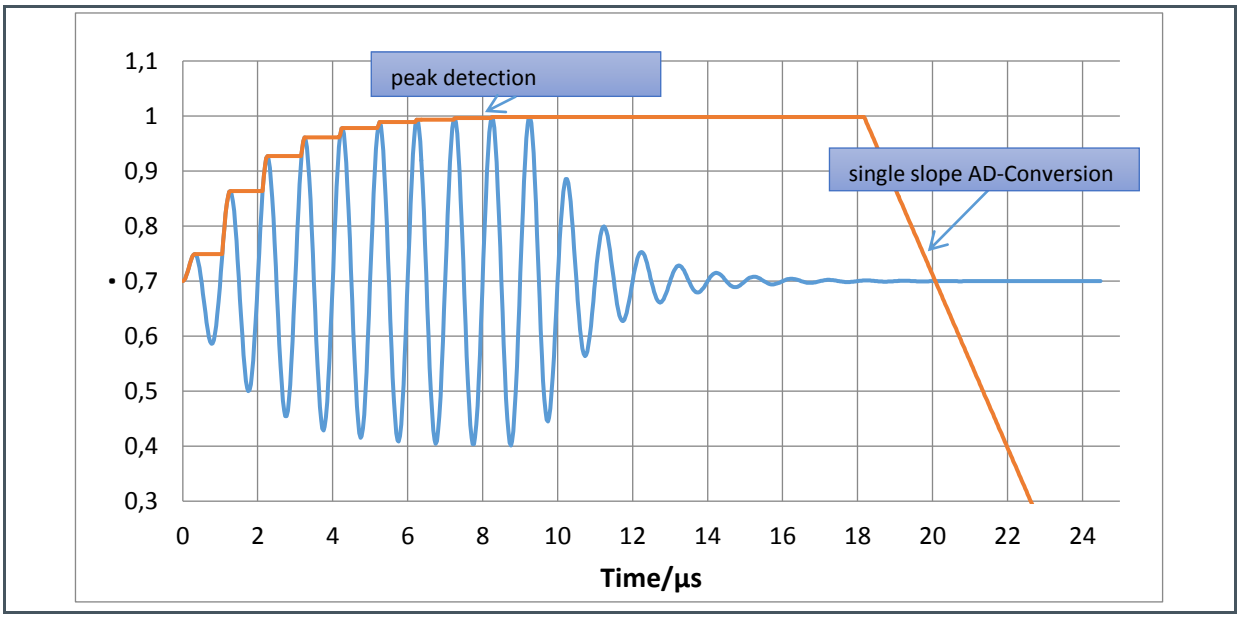
Figure 22:
Important Registers

Rehister	Address	Parameter
CR_GP_CTRL	0x0C2	**GP_CTRL**: This register should be configured to 0x4C0C00CC to connect ultrasonic signals to GPIOs
CR_USM_FRC	0x0C9	**TI_GM_MODE**: Enables gas meter mode **TI_PATH_EN[4:0]**: En-/disables internal paths of GP30's ultrasonic interface Depends on external application Please contact **ams** for further support **TI_ERA_EN**: Enables application with an external receive amplifier

2.2 Amplitude Measurement

A new feature in TDC-GP30 is a true amplitude measurement, which can be added to the TOF measurement. The first phase of the amplitude measurement is a peak detection which is performed while the receive burst is detected. After the end of the receive burst, the peak amplitude is measured by a single-slope AD-Conversion. Note that the peak detection can be configured to end at any hit after the first hit. This way it is possible to measure the peak amplitudes of receive burst hits during its rise individually. Of course this requires several separate measurements with different configurations.

Figure 23:
Amplitude Measurement



In Time Conversion Mode the result is given as raw time measurement data. From Frontend Data Buffer following raw data has to be read:

- **FDB_US_AM_U** ≡ AM_{Up} [ns]
- **FDB_US_AM_D** ≡ AM_{Down} [ns]
- **FDB_US_AMC_VH** ≡ AMC_H [ns]
- **FDB_US_AMC_VL** ≡ AMC_L [ns]

Those time data are converted into voltage by means of the following formulas.

Calculating the amplitude in mV:

$$V_{Up}[mV] = AMC_{Gradient}\left[\frac{mV}{ns}\right] * AM_{Up}[ns] - AMC_{Offset}[mV],$$

$$V_{Down}[mV] = AMC_{Gradient}\left[\frac{mV}{ns}\right] * AM_{Down}[ns] - AMC_{Offset}[mV],$$

with

- V_{CAL} (typ) = $V_{REF}/2$ = 350 mV, V_{REF} = internal reference
- $AMC_{Gradient}\left[\frac{mV}{ns}\right] = \frac{V_{CAL}[mV]}{AMC_H[ns]-AMC_L[ns]}$;
- $AMC_{Offset}[mV] = (2 * AMC_L[ns] - AMC_H[ns]) * AMC_{Gradient}\left[\frac{mV}{ns}\right]$

The amplitude measurement should end before the start of the TOF measurement:

AM_PD_END <= **TOF_START_HIT_NO**

Figure 24:
Important Registers

Register	Address	Parameter
CR_USM_AM	0x0CB	**AM_RATE**: Defines repetition rate for amplitude measurement **AM_MODE**: Defines moment when peak detection starts **AM_PD_END**: Defines number of waves when peak detection stops **AM_PD_WIN**: Defines peak detection window **AMC_RATE**: Defines repetition rate for amplitude measurement calibration

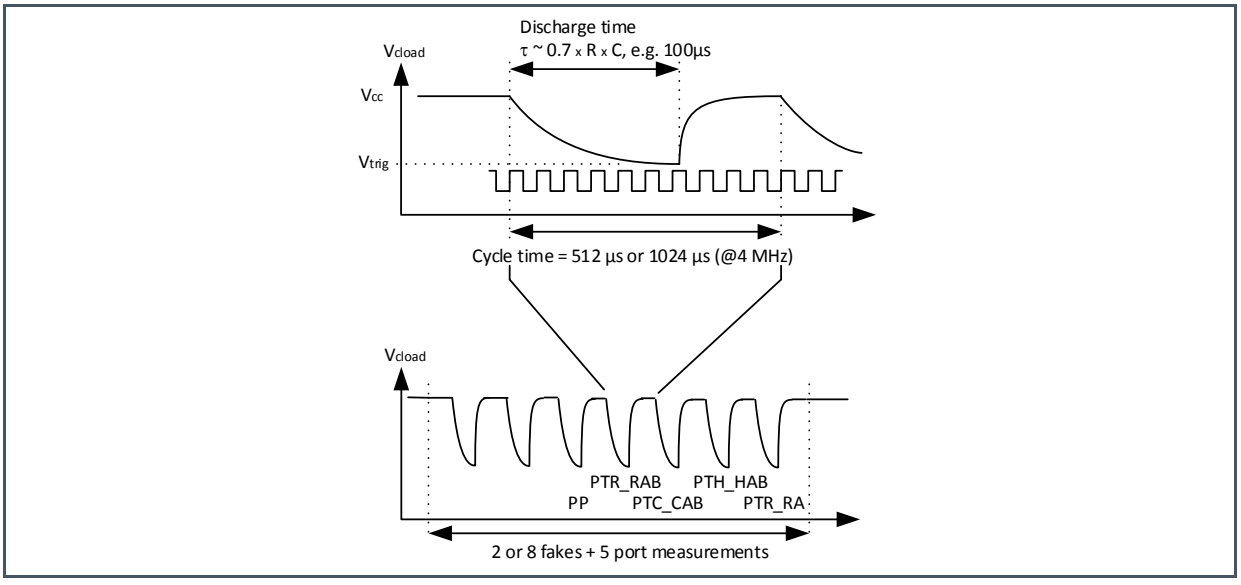
3 Temperature Measurement

In addition to the TOF measurement for flow metering, the GP30 has a highly accurate interface for sensor temperature measurements. This interface is suitable for various types of resistive temperature sensors. It can also be used for any other kind of resistive sensor.

3.1 Technical Properties of the Temperature Interface

The temperature interface performs resistance measurements by discharging a capacitor, which was loaded to the supply voltage VCC, over the unknown resistor network, down to some fixed comparison voltage. Each temperature interface pin contains a double switch which connects this pin to Cload (the temperature measurement load capacitor on pin CLOAD). Through different switch settings, all measurements needed are done in a well-controlled measurement sequence. The measurement sequences are hard-coded for 2-wire and 4-wire sensor case. Details on sequence configuration are discussed in subsequent sections. The following figure gives a schematic example of a 2-wire measurement sequence.

Figure 25:
Discharge Time Measurement



3.1.1 Measurement Range and Selection of Load Capacitor

One discharge cycle takes a time of about

$$\tau = 0.7 * R * C_{load},$$

for example 70 µs for a 1 kΩ resistor R and a 100 nF capacitor C_{load}.

The value of Cload and the range of R has to be chosen such that actual measurement times are between 10 µs (limited by the internal TDC calibration time) and the discharge cycle time minus the capacitor recharge time. The discharge cycle time tdct can be chosen to 512 µs or 1024 µs, respectively (**TM_DCH_SEL** in **CR_TM** set to 0 or 1, respectively). The capacitor recharge time can be estimated as

$$\tau_{re} = 10 * 10\,\Omega * C_{load},$$

for highest accuracy permit 3*τ_{re}. Typically Cload = 100 nF is used, which permits to measure resistances between 142 Ω and about 5 kΩ (maximal measurement time 502 µs).

3.1.2 Considerations On Measurement Accuracy

The short times of the measurements, corresponding to high signal frequencies, have to be considered when using the temperature interface:

- Long lines can be problematic through their parasitic inductance and capacitance. When using lines in the range of meters or even above, the quality of the measurements must be checked carefully.
- In addition, long lines may introduce problems through coupled noise and EMI. It is possible to reduce such problems by using filtering elements like ferrites or small capacitors, but the possible influence of such filters on the measurement has to be considered For details on that topic, refer to the **ams** Application Note 025: PSØ81: EMI Countermeasures for a Digital Load Cell
- It is also recommended to use a capacitor of COG or other class-1-type. For example, X7R-types can suffer from memory effects and, of course, from high temperature coefficients, which may reduce measurement accuracy dramatically.

It is in any case recommended to control the quality of such resistance measurements in your actual measurement environment.

Measurement accuracy is achieved by a suitable calibration and based on the measurement of a well-known reference resistor. The equation for τ, given in the former section, has no guaranteed accuracy and can only be used for resistance estimations. Still, high measurement accuracy is reached by comparisons, making use of the high short-term repeatability and linearity of the single resistance measurements. Thus, a typical measurement sequence includes the measurement of a well-known reference resistor, such that actual resistance values can be calculated from the ratios of measured times τ to τ_{REF}, the time result for the reference resistor measurement. The accuracy of such measurements depends on the following factors:

- Absolute accuracy, temperature dependence and aging of the reference resistor is of course fully traced to the measurement result. For high quality measurements, use a reference resistor with low tolerance and low TC.
- Offset line and switch resistances. Such offset values are unavoidable, their removal requires additional calibration measurements. The following sections describe how such measurements are setup and used in case of 2-wire and 4-wire measurements. Switch resistances also add to the reference resistor measurement, typical values for switch resistances in GP30 are below 10 Ohms.
- Linearity and repeatability of measurements: Due to a stable comparison voltage, linearity can be considered ideal. The good short-term stability of the temperature interface can even be further improved:
  - All measurements needed are done in a well-controlled measurement sequence. The measurement sequences are hard-coded, but can be configured as described below.
  - 2 or 8 fake measurements (configurable) are done before each measurement sequence. This makes sure that each relevant measurement starts in a similar condition (all measurement taken into account did have at least two similar measurements before). The decision about 2 or 8 fake measurements depends mainly on the quality of Cload, usually 2 should be enough.
  - The measurement sequence can be repeated (configurable) after a fixed time of, for example, 1.5 times the base frequency period. Setting the base frequency to the local mains frequency (50 or 60 Hz) results in suppression of power line noise in temperature measurements.
  - The repeated measurement sequence can be configured to be in reverse order. This removes any linear deviation trend that may appear during measurements, for example changing sensor temperature during a measurement cycle.
- Finally, the noise of the TDC measurements, which are utilized for the temperature ports as well, set an absolute limit on measurement accuracy. With a typical single-shot peak noise level of about ±2.5ns, the equivalent peak noise in measured resistance ratios can be estimated as ±0.001 % of full scale values (assuming 500µs maximal measurement time). Note that this is an absolute error, such that the relative error increases for low measurement times or low resistances, respectively.

3.1.3 Measurement Run Time and Current Consumption

The total runtime tti of a temperature interface measurement sequence depends on the chosen configuration and can be calculated as

$$t_{ti} = t_{HSO} + t_{dct} * \left(n_{fake} + n_{meas}\ \right) + t_{pause}$$

Here, n_{meas} is the number of actual resistance measurements (4 in 2-wire 2-port case, 5 for 2-wire 3-port, always 14 for 4-wire case and 3 for internal measurement). t_{HSO} is the configured HSO settling time, t_{dct} the chosen discharge cycle time (512 or 1024 µs), n_{fake} the number of fake measurements (2 or 8) and t_{pause} the configured pause time (could even be 0, then the measurement sequence is not repeated).

Example:
The total runtime of a 4-wire 3-port measurement with 2 fake measurements, at 135 µs HSO settling time and 512 µs discharge cycle time and with 30 ms pause time is 38.33 ms. If a firmware is used to evaluate the results, about 0.5 ms runtime is added. It should be clear that the total measurement time is dominated by the pause time. Note that the pause time starts together with the first measurement sequence, such that the first measurement sequence takes place during pause time.

Of course, the average current consumption depends strongly on the total runtime tti as well as on the temperature measurement rate or its frequency f_{tm}, respectively. A reasonable estimation of the additional average current consumption for the temperature measurement interface I_{tm} is

$$I_{tm} = 0.6\ \mu A * \left(n_{fake} + n_{meas}\ \right) * f_{tm} * \frac{C_{load}}{100\ \mathrm{nF}}$$

This formula gives an upper limit for repeated measurement sequence and 512 µs discharge cycle time, the value for 1024 µs is about 15% higher. At only one single measurement sequence

(t_{pause} = 0), the actual current consumption is half of the calculated value.

Example:
Average current consumption for external 2-wire 3-port measurements with 2 fake measurements and repeated (t_{pause} ≠ 0), at a cycle time of 125ms and **TM_RATE** = 240

(f_{tm} = 1/30 Hz) is estimated to 0.14 µA for 512 µs discharge cycle time or 0.161 µA for 1024 µs. At (t_{pause} = 0), the result is 0.07 µA for 512 µs. Note that the current consumption does not depend on the measured resistance. Note also that a low repetition rate, as in the given example, the current consumption comes in shape of a peak at the temperature measurement frequency. It is recommended to use a 100 µF blocking capacitor on supply voltage for high quality temperature measurements.

3.1.4 Temperature Measurement Interface Error Messages

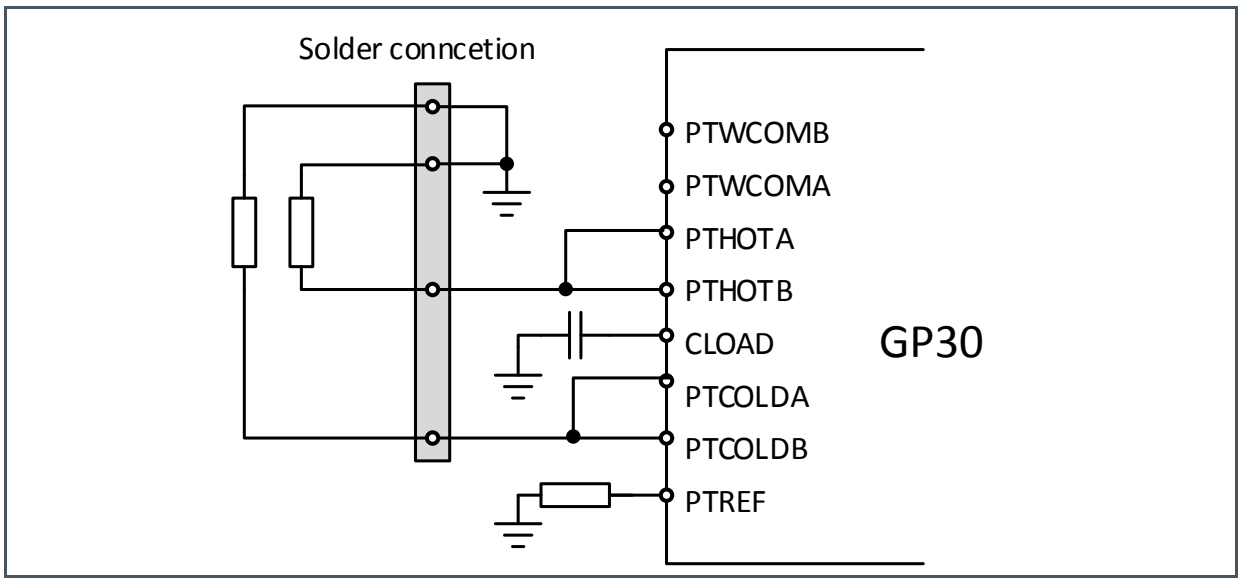
The temperature interface generates error messages, which can be read from register **SRR_ERR_FLAG**. There are three different error flags that may be set by the temperature interface:

- **EF_TM_SQC_TMO** (bit 8): Temperature Sequence Timeout. This flag is set when the first temperature measurement sequence did not finish before the end of the configured pause time. Note that this flag must be ignored when no second measurement block is configured, since then there is no pause time and the flag is always set.
- **EF_TM_SC_ERR** (bit 4): Temperature Measurement Short Circuit. This flag is set when the load capacitor is discharged very quickly, such that within the first 1 µs of the measurement the capacitor voltage drops below VCC /2. This indicates a too low resistance, which typically happens in case of a short circuit of a sensor.
- **EF_TM_OC_ERR** (bit 3): Temperature Measurement Open Circuit. This flag is set when the load capacitor is not discharged to the comparison voltage level during a measurement cycle at all. This indicates a too high resistance, which typically happens in case of an open circuit, for example a loose sensor cable.

3.2 Two-Wire Mode

In two-wire mode, the temperature interface does a sequence of resistor measurements in different configurations to provide a set of measurements for accurate determination of one or two resistors. The sensors are connected to the chip as drawn in the next figure:

Figure 26:
2-Wire T-Measurement



The sequence contains the following time measurements:

- t_{PP}: Cload discharged over cold sensor (plus two parallel double switches) and reference resistor (plus one double switch) in parallel.
- t_{RAB}: Cload discharged over reference resistor, connected over one double switch.
- t_{CAB}: Cload discharged over cold sensor resistor, connected over two parallel double switches (both pins PTCOLDA and PTCOLDB connected to cold sensor).
- t_{HAB}: Cload discharged over hot sensor resistor, connected over two parallel double switches (both pins PTHOTA and PTHOTB connected to hot sensor). This measurement can be configured optionally.
- t_{RA}: Cload discharged over reference resistor, connected over a single switch (only half double switch closed).

As discussed above, this measurement sequence may be repeated, e.g. in reverse order, and the whole sequence may again be repeated after a configurable pause time.

Results of these five measurements are stored in the frontend data buffer.

Figure 27:
FDB with T-Measurement

RAA Address	Name	Description
0x080	***FDB_TM_PP_M1***	Schmitt trigger delay compensation value
0x081	***FDB_TM_PTR_RAB_M1***	PT Ref: Impedance value
0x082	***FDB_TM_PTC_CAB_M1***	PT Cold: Impedance value
0x083	***FDB_TM_PTH_HAB_M1***	PT Hot: Impedance value
0x084	***FDB_TM_PTR_RA_M1***	PT Ref: 1st Rds(on) correction value
0x085	***FDB_TM_PP_M2***	Schmitt trigger delay compensation value
0x086	***FDB_TM_PTR_RAB_M2***	PT Ref: Impedance value
0x087	***FDB_TM_PTC_CAB_M2***	PT Cold: Impedance value
0x088	***FDB_TM_PTH_HAB_M2***	PT Hot: Impedance value
0x089	***FDB_TM_PTR_RA_M2***	PT Ref: 1st Rds(on) correction value

Values with names ending in **M2** come from the repeated measurements. They remain unchanged when no second measurement is done (**TM_PAUSE** = 0). Letters before the measurement number indicate active port (**R**ef., **C**old and **H**ot, pin **A** and/or **B**;). The raw results given here are raw TDC values (to be converted to actual times by multiplying with t_{HSO}/216). But since in the final calculation only ratios of values will be used, no further formatting of the raw results is needed.

Measurements number 2. to 4. (t_{RAB} ,t_{CAB} "and" t_{HAB}) correspond to the total resistance values of the measured network, including internal switch resistances. To remove the influence of switch resistances and comparator delay, measurements 1. and 5. (t_{PP} "and" t_{RA}) should be used according to the following equations:

$R_{ds(on)}$ correction (the correction of switch resistances) $t_{RO} = t_{RA}$

Schmitt trigger delay compensation $\Delta t = 2t_{PP} - 2\frac{t_{CAB}\, t_{RAB}}{t_{CAB}+t_{RAB}}$

Note that the Schmitt trigger delay compensation requires a measurement of the cold sensor. In case one sensor may be optional, always use the hot sensor for the optional one.

Reference resistor time $t_R = t_{RAB} - t_{RO} - \Delta t$

Cold sensor time $t_C = t_{CAB} - {}^{t_{RO}}/_{2} - \Delta t$

Hot sensor (3-port case) time $t_H = t_{HAB} - {}^{t_{RO}}/_{2} - \Delta t$

With the known reference resistor value R_REF we then get the

Cold sensor resistance: $R_C = R_{REF}\,\frac{t_C}{t_R}$

Hot sensor resistance (3-port case): $R_H = R_{REF} \frac{t_H}{t_R}$

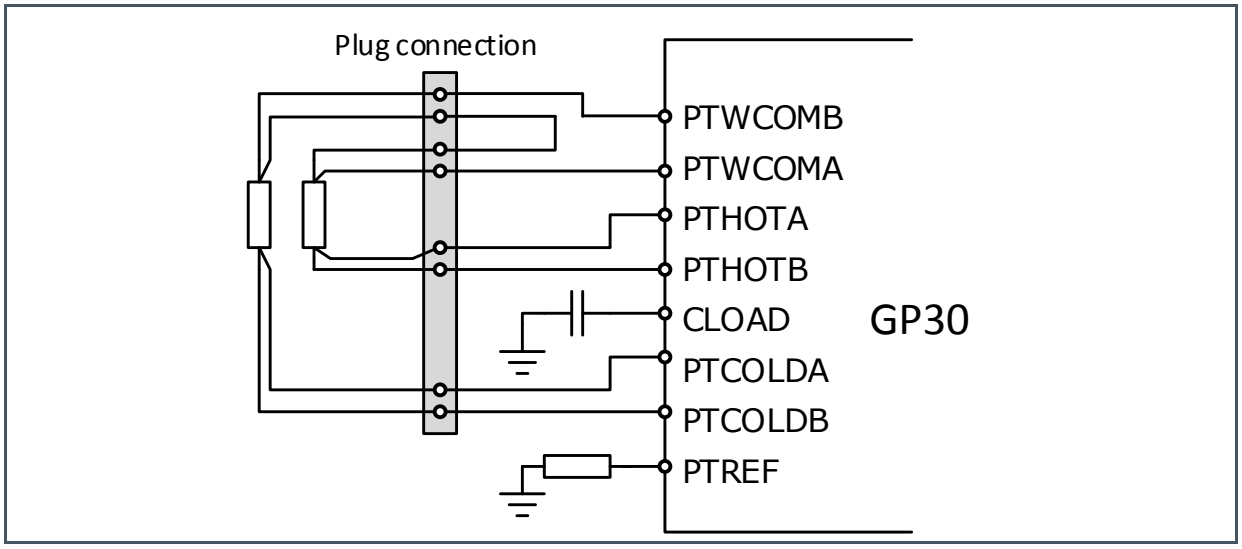
This calculation assumes that all double switches are identical, and that the measurements are linear and repeatable (see details in section 3.1). Under these conditions, the sensor network resistances are measured to the accuracy of the reference resistor, with an uncertainty through added noise of ±0.001 % of full scale. Additional line resistances in the sensor networks cannot be calibrated out by 2-Wire measurements. If the line resistances are known from different measurements, they may simply be subtracted from the result in a separate calibration. The calculation of temperatures from resistance values is done as usual through the sensor's characteristic T(R) curve.

Note that the calibration measurement for comparator delay uses the cold sensor. If the cold sensor path is unusable, for example due to some damage, also the hot sensor path cannot be fully calibrated. So, if only one sensor is used, always use the cold sensor pins, and if one sensor result is needed with high accuracy, even in case the other sensor may be damaged, connect the more important sensor to the cold sensor ports.

3.3 Four-Wire Mode

In 4-wire mode, the temperature interface does a sequence of resistor measurements in different configurations to provide a set of measurements for accurate determination of the sensor resistance of one or two 4-wire sensors. The sensors should be connected to the chip as drawn in then following figure:

Figure 28:
4-Wire T-Measurement



In case only one 4-wire sensor is used, connect its two ground cables separately to PTWCOMA and PTWCOMB.

The sequence contains the following time measurements. All inactive pin pairs PTCOLDA/PTCOLDB or PTHOTA/PTHOTB are usually grounded in these measurements (configurable by **TM_PORT_MODE** in **CR_TM**, see the data sheet, DB_GP30_Vol1). Unless otherwise noted, the switched ground pins PTWCOMA and PTWCOMB are also both grounded:

- t_{PP}: Cload discharged over cold sensor (plus two parallel double switches) and reference resistor (plus one double switch) in parallel.
- t_{RAB}: Cload discharged over reference resistor, connected over one double switch.
- t_{CAB}: Cload discharged over cold sensor resistor, connected over two parallel double switches.
- t_{HAB}: Cload discharged over hot sensor resistor, connected over two parallel double switches.
- t_{RA}: Cload discharged over reference resistor, connected over a single switch (only half double switch closed).
- t_{RB}: Cload discharged over reference resistor, connected over the other single switch (other half double switch closed).
- t_{CA}: Cload discharged over cold sensor resistor, connected over double switch “A”.
- t_{CB}: Cload discharged over cold sensor resistor, connected over double switch “B”.
- t_{AC}: Cload discharged over cold sensor resistor, connected over two parallel double switches (both pins PTHOTA and PTHOTB active). Only ground pin PTWCOMA is switched.
- t_{BC}: Cload discharged over cold sensor resistor, connected over two parallel double switches (both pins PTHOTA and PTHOTB active). Only ground pin PTWCOMB is switched
- t_{HA}: Cload discharged over hot sensor resistor, connected over double switch “A”.
- t_{HB}: Cload discharged over hot sensor resistor, connected over double switch “B”.
- t_{AH}: Cload discharged over hot sensor resistor, connected over two parallel double switches (both pins PTHOTA and PTHOTB active). Only ground pin PTWCOMA is switched
- t_{BH}: Cload discharged over hot sensor resistor, connected over two parallel double switches (both pins PTHOTA and PTHOTB active). Only ground pin PTWCOMB is switched.

Results of these fourteen measurements are stored in the frontend data buffer.

Figure 29:
FDB with 4-Wire T-Measurement

RAA Address	Name	Description
0x080	***FDB_TM_PP_M1***	Schmitt trigger delay compensation value
0x081	***FDB_TM_PTR_RAB_M1***	PT Ref: Impedance value
0x082	***FDB_TM_PTC_CAB_M1***	PT Cold: Impedance value
0x083	***FDB_TM_PTH_HAB_M1***	PT Hot: Impedance value
0x084	***FDB_TM_PTR_RA_M1***	PT Ref: 1st Rds(on) correction value
0x085	***FDB_TM_PP_M2***	Schmitt trigger delay compensation value
0x086	***FDB_TM_PTR_RAB_M2***	PT Ref: Impedance value
0x087	***FDB_TM_PTC_CAB_M2***	PT Cold: Impedance value
0x088	***FDB_TM_PTH_HAB_M2***	PT Hot: Impedance value
0x089	***FDB_TM_PTR_RA_M2***	PT Ref: 1st Rds(on) correction value
0x08A	***FDB_TM_PTR_4W_RB_M1***	PT Ref: 2nd Rds(on) correction value

0x08B	***FDB_TM_PTC_4W_CA_M1***	PT Cold: 1st Rds(on) correction value
0x08C	***FDB_TM_PTC_4W_CB_M1***	PT Cold: 2nd Rds(on) correction value
0x08D	***FDB_TM_PTC_4W_AC_M1***	PT Cold: 3rd Rds(on) correction value
0x08E	***FDB_TM_PTC_4W_BC_M1***	PT Cold: 4th Rds(on) correction value
0x08F	***FDB_TM_PTH_4W_HA_M1***	PT Hot: 1st Rds(on) correction value
0x090	***FDB_TM_PTH_4W_HB_M1***	PT Hot: 2nd Rds(on) correction value
0x091	***FDB_TM_PTH_4W_AH_M1***	PT Hot: 3rd Rds(on) correction value
0x092	***FDB_TM_PTH_4W_BH_M1***	PT Hot: 4th Rds(on) correction value
0x093	***FDB_TM_PTR_4W_RB_M2***	PT Ref: 2nd Rds(on) correction value
0x094	***FDB_TM_PTC_4W_CA_M2***	PT Cold: 1st Rds(on) correction value
0x095	***FDB_TM_PTC_4W_CB_M2***	PT Cold: 2nd Rds(on) correction value
0x096	***FDB_TM_PTC_4W_AC_M2***	PT Cold: 3rd Rds(on) correction value
0x097	***FDB_TM_PTC_4W_BC_M2***	PT Cold: 4th Rds(on) correction value
0x098	***FDB_TM_PTH_4W_HA_M2***	PT Hot: 1st Rds(on) correction value
0x099	***FDB_TM_PTH_4W_HB_M2***	PT Hot: 2nd Rds(on) correction value
0x09A	***FDB_TM_PTH_4W_AH_M2***	PT Hot: 3rd Rds(on) correction value
0x09B	***FDB_TM_PTH_4W_BH_M2***	PT Hot: 4th Rds(on) correction value

Values with names ending in **M2** come from the repeated measurements. They remain unchanged when no second measurement is done (**TM_PAUSE** = 0). Letters before the measurement number indicate active port (**R**ef., **C**old and **H**ot, **A** or **B**; preceding **A** or **B** means ground port switched).

In contrast to the 2-Wire measurement, every single switch resistance of different pins can be calculated from these measurements. This is important, since every pin may be connected to the sensor over a different line resistance. There is no assumption about similar switches or similar line lengths. This makes this measurement method particularly suitable for high-accuracy measurements using connectors or other network elements that may suffer from aging.

The calibration calculation system for resolving sensor resistances from the measurement results is incorporated in **ams** firmware for GP30. Of course any customized calculation system or wiring can be used, based on switch settings as given above.

The four-wire mode can also be used to measure up to four separate resistor values in 2-wire quality. In this case, each of the (up to) four resistor networks should be connected to one of pins PTCOLDA, PTCOLDB, PTHOTA and PTHOTB, and their other terminal directly grounded. In simplest case, this yields four resistance measurements in 2-wire quality, using the additional measurements in a similar way as described in the section 3.2 on 2-wire measurements.

In this simple case, half of the measurements are obsolete. They can, however, not be switched off due to the hard-coded measurement sequence. More advanced methods can be used to improve the result's accuracy when the other "obsolete" measurements are taken into account. The corresponding calculations depend on the particular network. For example in case of a half-bridge or full-bridge

network, benefit may be taken from the two switched ground pins PTWCOMA and PTWCOMB which take different switch states in four of the measurements. The various possibilities of such advanced methods are beyond the scope of this document.

3.4 Internal Temperature Measurement

A simple temperature measurement is also possible through a build-in resistive sensor. This measurement uses the same hardware as the external temperature interface, and the measured results are stored in the same cells as external reference and cold resistor results. Consequently, external or internal temperature measurements are mutually exclusive during one measurement cycle. However, GP30 can be configured to toggle between external and internal sensor from cycle to cycle, such that both types of measurements can be used alternatingly if desired (set **TM_MODE** in **CR_TM** to 1X, see data sheet DB_GP30_Vol1).

The measurement sequence contains the following time measurements (**M1**):

- t_{PP}: C_{load} discharged over cold sensor (plus two parallel double switches) and reference resistor (plus one double switch) in parallel.
- t_{RAB}: C_{load} discharged over reference resistor, connected over one double switch.
- t_{CAB}: C_{load} discharged over cold sensor resistor, connected over two parallel double switches (both pins PTCOLDA and PTCOLDB connected to cold sensor).

As discussed above, this measurement sequence may be repeated (M2), e.g. in reverse order, and the whole sequence may again be repeated after a configurable pause time.

Results of these three or six measurements are stored in the frontend data buffer.

Figure 30:
FDB with Internal T-Measurement

RAA Address	Name	Description
0x080	***FDB_TM_PP_M1***	Schmitt trigger delay compensation value
0x081	***FDB_TM_PTR_RAB_M1***	PT Ref: Impedance value
0x082	***FDB_TM_PTC_CAB_M1***	PT Cold: Impedance value
0x085	***FDB_TM_PP_M2***	Schmitt trigger delay compensation value
0x086	***FDB_TM_PTR_RAB_M2***	PT Ref: Impedance value
0x087	***FDB_TM_PTC_CAB_M2***	PT Cold: Impedance value

The internal temperature measurement utilizes an internal reference resistor with a nominal value of 1.265 kΩ at room temperature and a temperature coefficient of -0.1 Ω/K, and a sensor resistor with the same nominal resistance value, but a different temperature coefficient of 3.7 Ω/K. Due to chip tolerances, this measurement will not be very accurate. Under the assumption that the combined temperature coefficient of the resistance ratios is known as 3.8 Ω /K, a simple calculation can be done when a measurement at known chip temperature is performed:

$$T = \left(\frac{t_{CAB}}{t_{RAB}} - \frac{t_{CAB}(T_0)}{t_{RAB}(T_0)}\right) * \frac{1.265\text{k}\Omega}{3.8\Omega/\text{K}} + T_0$$

Here T_0 is the temperature at the calibration measurement, typically room temperature, and t_{CAB} and t_{RAB} are the times measured for cold resistor in cell 0x082: **FDB_TM_PTC_CAB_M1** and for the reference resistor in cell 0x081: **FDB_TM_PTC_RAB_M1**. Due to chip tolerances, at least one calibration measurement at some temperature T_0 is recommended. It is actually sufficient to assume

$$\frac{t_{CAB}(T_0)}{t_{RAB}(T_0)} = 1$$

and to adjust the constant T_0 in the upper equation for the correct result.

More elaborate calibrations and calculations are possible. For example the Schmitt trigger delay compensation could be done. However, the internal temperature sensor is not accurate enough to justify that effort. It would also be possible to find an individual value for the internal sensor's temperature coefficient, but that would require a measurement at different temperature. Nevertheless, with the simple calibration described above the internal temperature sensor can reach accuracies down to a few centigrade, which is good enough for some applications.

3.5 Optional Calibration Steps and Relation between Resistance and Temperature

The two main measurement sequences discussed above include a number of calibration measurements for correction of some typical measurement deviations. In case of 4-Wire measurements, no further steps to improve the resistance measurement accuracy need to be done, while in case of 2-Wire measurements an optional calibration can be used to further increase measurement accuracy. In general, a known resistance offset in 2-Wire case can be subtracted from the measured result. Gaining knowledge about this offset resistance requires optional calibration measurements, for example measurements at a known temperature with known resistance values.

Calculation of temperature values from measured resistance values requires calibration or at least knowledge of a relation between the sensor resistance and sensor temperature. In case of a standard PT1000 or PT500 sensor, the relation between resistance and temperature is standardized. An approximation for T(R) can be derived which features an accuracy of 3mK within 0°C and 100°C:

$\frac{T}{^\circ C} = \boldsymbol{C_2} * \left(\frac{R}{R_0}\right)^2 + \boldsymbol{C_1} * \left(\frac{R}{R_0}\right) + \boldsymbol{C_0}$ where $\boldsymbol{C_2 = 10.115\ C_1 = 235.57,\ C_0 = -245.683}$

4 CPU Handling

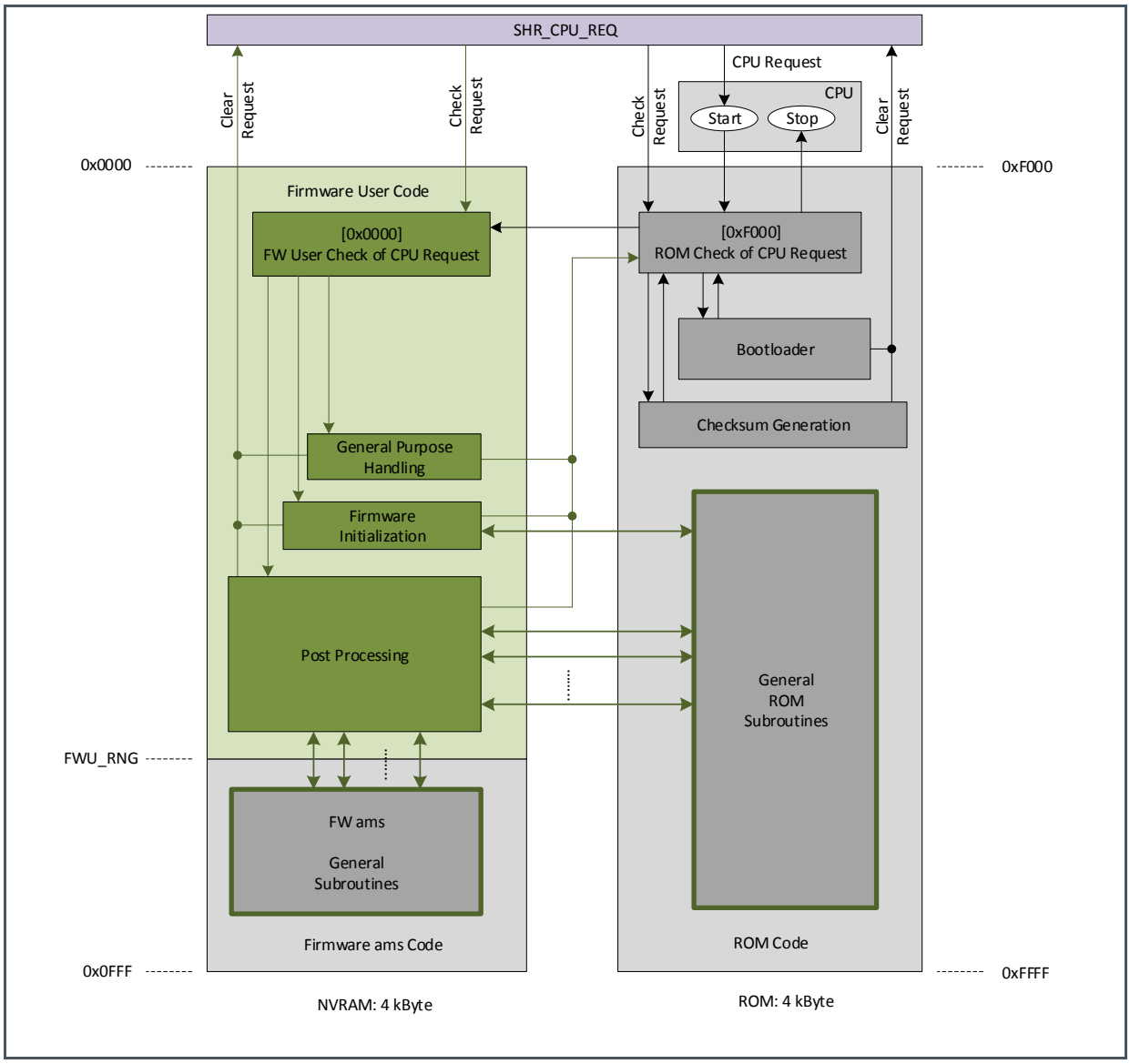
CPU handling is performed via register **SHR_CPU_REQ**. There, following requests are possible to start execution of program code in CPU:

- Bootloader
- Checksum Generation
- Firmware Initialization
- Post Processing
- General Purpose Handling

In general, sending any of these requests by setting the corresponding bit in **SHR_CPU_REQ** lets the chip start the CPU at the appropriate position within the task sequencer cycle. Firmware operation then always starts within the ROM code by checking the request. Bootloader and checksum generation requests are handled directly in ROM routines, while for Firmware initialization, post processing and General purpose handling the firmware user code is started, always at the program start.

After the request is processed, the firmware must clear it in **SHR_CPU_REQ** (or simply clear the whole register after all is done), else the request remains.

Figure 31:
CPU and Code Structure



Above figure sketches the different sequences that could be triggered by a CPU request. They are explained more in detail in the subsequent sections. For more details on CPU and **ams** firmware, please refer to manuals GP30_Vol2 and GP30_Vol4.

Program code in green color can be defined and programmed by customers, whereby public subroutines in FW ams code or some ROM code can also be used by customer. In case the unmodified **ams** firmware is used, no programming on customer side is needed.

4.1 Check of CPU Request

In case any of the requests is set in **SHR_CPU_REQ** the CPU starts first in ROM code with a check of request type.

If requests for bootloader or checksum generation are set then these requests will be served directly in ROM code.

For the other 3 CPU requests, the check has to be processed in FW code, where users can define at which FW code location the requests should be served.

For more information on firmware development please refer to “Firmware User Guide GP30_Vol4”.

Figure 32:
CPU Request Handling

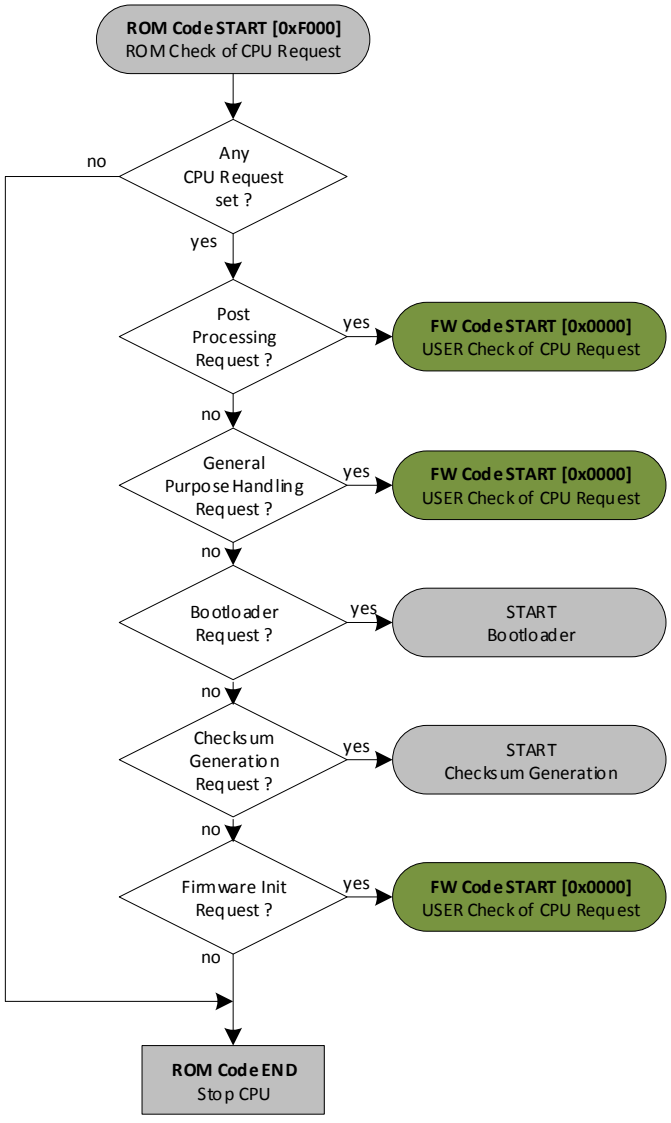


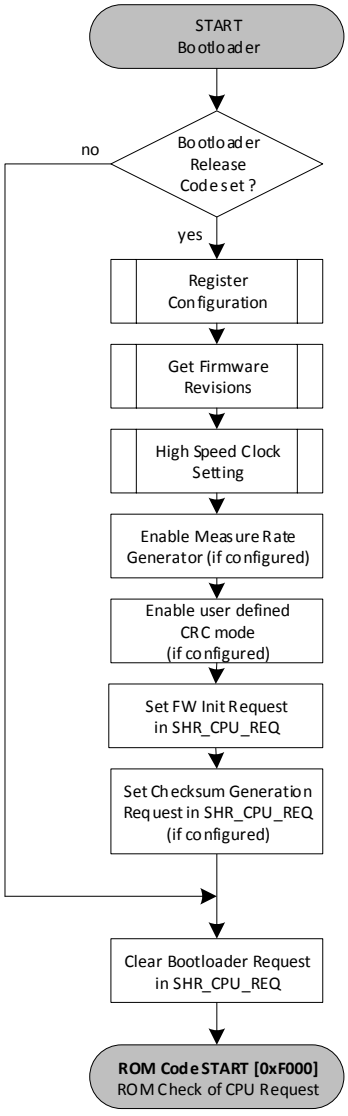
Figure 33:
Important Registers

Register	Address	Parameter
SHR_CPU_REQ	0x0DC	CPU Requests

4.2 Bootloader

The bootloader is always requested after a system reset or a system INIT occurred. However, bootloader actions are only performed if the bootloader release code is set.

Figure 34:
Bootloader Actions



Bootloader actions are:

- “Register Configuration” which transfer configuration data to register area
- “Get Firmware Revisions” which set FW revisions in register **SRR_FWU_REV** & **SRR_FWA_REV**
- “High Speed Clock Setting” which transfer the **HS_CLK_SEL** bit from **CR_CPM** to **SHR_RC** from which high speed clock setting is controlled
- Enabling Measure Rate Generator if **MR_CT** in **CR_MRG_TS** is configured to a value > 0
- Enabling User defined CRC mode, by transferring **UART_CRC_MODE** from **CR_UART** to **SHR_RC** from which CRC mode setting is controlled
- Setting “FW Init” request which is performed after bootloader sequence has been finished
- Setting “Checksum Generation” request, if **CPU_BLD_CS** is set in **CR_IEH**, which is directly performed after bootloader sequence finished

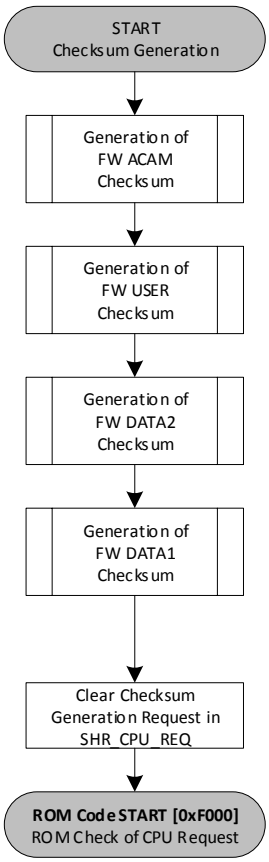
Finally, the bootloader clears its request in **SHR_CPU_REQ** and jumps back to ROM code for checking CPU requests.

Figure 35:
Important Registers

Register	Address	Parameter
BLD_RLS	0x17B	Release Code for bootloader 0xABCD_7654: Release code for bootload execution Others: Bootloader is not executed

4.3 Checksum Generation

Figure 36:
Checksum Generation



Checksum generation can be requested by remote command **RC_FW_CHKSUM** or, if con-figured, by bootloader. Note that the recommended code for register **CR_TRIM3** is 0x00250808 when **RC_FW_CHKSUM** should be used.

Then the checksums of all four FW areas are generated and compared to checksums which can be stored at the last four addresses of FW Data memory.

Finally, the checksum generation clears its request in **SHR_CPU_REQ** and jumps back to ROM code for checking CPU requests.

It is possible to calculate the checksums for all FW data and code areas outside the chip, when their data content is known. This is done by simply adding up all 8-bit words of code or data. Note that both FW code and data area are interpreted as 8-bit words for checksum generation, even though FW data is organized in 32-bit words. The FW ams code part is confidential, such that a customer cannot calculate its checksum independently. If required, this checksum must be generated by the chip as described above.

5 Remote Communication

Remote communication can be started within a task sequencer cycle, after frontend and post processing finished. TDC-GP30 signalizes this instant to a remote controller by

- Interrupt via pin INTN_DIR for SPI communication
- Interrupt message for UART communication

Generating an interrupt request for remote communication is served by TDC-GP30 in following ways:

- Interrupt is automatically sent with every task sequencer cycle by enabling **IRQ_EN_TSQ_FNS** in **CR_IEH**. This is typically used in time conversion mode to allow the remote controller reading raw measurement values from frontend data buffer after each measurement.
- Interrupt is controlled by firmware. Therefore **IRQ_EN_FW_S** in **CR_IEH** has to be set. Then a synchronous interrupt can be triggered by firmware by setting **FW_IRQ_S** in **SHR_EXC**. The interrupt will appear after post processing finished. Typically used in flow conversion mode where a remote communication need not be requested with every task sequencer cycle.

In case of a running firmware, it is possible to avoid permanent communication and to keep the external controller in sleep mode over long periods. Since the firmware can evaluate and store results, it does not need to communicate after each measurement. It can then be advantageous to let the external controller trigger communication: An external controller can send command **RC_COM_REQ** to TDC-GP30 to request a remote communication at an arbitrary time. This causes that **COM_REQ** will be set in register **SRR_MSC_STF**. By polling this status flag, the firmware is able to trigger remote communication as described above on request by the external controller.

IMPORTANT NOTE

A synchronous communication request can be issued at any time (asynchronously) by the remote command **RC_COM_REQ**. After sending this command, no further communication is permitted until GP30 replies with a synchronous interrupt at the beginning of its idle state. This interrupt has to be initiated by a firmware. The firmware in TDC-GP30-F01 does this. The interrupt is the start signal for synchronized communication between GP30 and the external controller. During this communication, the communication request has to be cleared by setting **COM_REQ_CLR** (bit 12) in register **SHR_EXC**, preferably by the firmware. Not clearing the communication request may cause uncontrolled subsequent interrupts. In TDC-GP30-F01 with firmware versions A1.A2.11.04 or higher, **COM_REQ** is reset by the firmware, and GP30 answers with a single interrupt.

Figure 37:
Important Registers

Register	Address	Parameter
CR_IEH	0x0C4	**IRQ_EN_TSQ_FNS**: Interrupt mask for Task Sequencer Finished **IRQ_EN_FW_S**: Interrupt mask for synchronized FW interrupt request

Register	Address	Parameter
SHR_EXC	0x0DD	**FW_IRQ_S**: Triggers synchronized FW interrupt request **COM_REQ_CLR**: Clears communication request by remote controller
SRR_MSC_STF	0x0EA	**COM_REQ**: Communication request by remote controller

5.1 Getting Measurement Results

The following sections define the recommended sequences via remote interface for SPI.

The defined opcodes & data are in principle the same for UART, but the different protocol characteristics for UART communication need to be considered (Little Endian, block length byte, CRC, acknowledge).

5.1.1 Time Conversion Mode

In time conversion mode the availability of new measurement data is typically signalized by interrupt automatically after frontend processing

Figure 38:
Time Conversion Mode

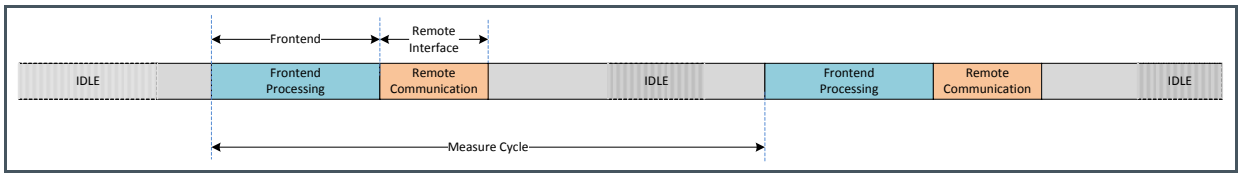


Figure 39:
Reading Results

Step	Description	Opcodes & Data	Sequence
1	Check on interrupt pin		Wait for interrupt
1a	Optionally read current task sequencer time from **SRR_TS_TIME** to check how much time is left for remote communication	0x7A 0xE9 read data	
2	Read **SRR_ERR_FLAG** to check if any error occurred during last measurement cycle	0x7A 0xE1 read data	Get Status
3	Read **SRR_FEP_STF** to check which measurement has been updated in last measure cycle	0x7A 0xE2 read data	

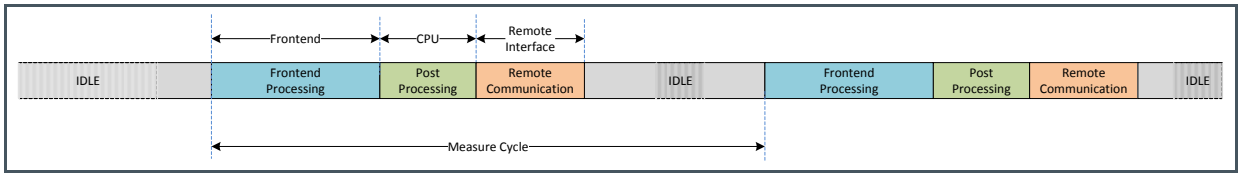
Step	Description	Opcodes & Data	Sequence
4	Dependent on error flags set in **SRR_ERR_FLAG** and updated measurements set in **SRR_FEP_STF**, read the measurement results out of the frontend data buffer	0x7A 0x80 read data …….	Get results
5	Clear interrupt flag, error flag & frontend status flag register by writing code to **SHR_EXC**	0x5A 0xDD 0x00000007	Completion

Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x7A for reading from RAM or register area, 0x7B for reading from FWD in NVRAM)

5.1.2 Flow Meter Mode

In flow meter mode the availability of calculated data is typically signalized by interrupt controlled by firmware and set after post processing

Figure 40:
Flow Meter Mode



In the following sequence some register names in RAM area and their appropriate addresses are referenced to **ams** firmware. For a more detailed description of **ams** firmware and its flow metering routines, please refer to “TDC-GP30 Datasheet Vol. 4”.

Figure 41:
Reading Results

Step	Description	Opcodes & Data	Sequence
1	Check on interrupt pin		Wait for interrupt
1a	Optionally read current task sequencer time from **SRR_TS_TIME** to check how much time is left for remote communication	0x7A 0xE9 read data	
2	Read **RAM_R_FW_ERR_FLAG** to check if any error has been detected and set by firmware	0x7A 0x27 read data	Get Status
3	Dependent on error flags set in **SRR_ERR_FLAG**, read the measurement results out of the RAM results area of FW	0x7A 0x00 read data …….	Get results
4	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion

Step	Description	Opcodes & Data	Sequence
5	Clear FW error flags by writing 0x00000000 to **RAM_R_FW_ERR_FLAG**	0x5A 0x27 0x00000000	

Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x7A for reading from RAM or register area, 0x7B for reading from FWD in NVRAM)

Another way to get results from low meter mode is to use the pulse interface, which is described in the section 5.3.

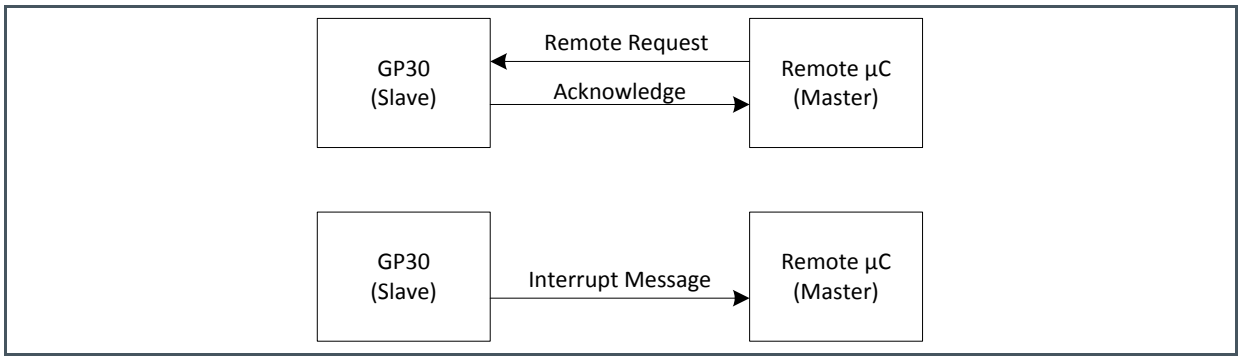
5.2 UART Communication

5.2.1 Basic Modes

For UART communication the TDC-GP30 typically acts as a slave, which gets a request from the remote µC. TDC-GP30 itself confirms each request by sending an acknowledge.

As an exception, TDC-GP30 is also able to send an automatic interrupt message (e.g. to initiate a request performed by remote µC to read out measuring results). Optionally the interrupt message can also include measuring results, such that no subsequent remote request is needed to get the results.

Figure 42:
UART Modes



The default and initial baud rate of TDC-GP30 is 4800 Baud and a baud rate change is typically requested by the remote µC.

As a further option, TDC-GP30 can be configured to send the interrupt message automatically with a high baud rate.

Additionally, TDC-GP30 has the option to send a wakeup byte (0x00) at the beginning of an interrupt message to allow the remote µC to wake up from sleep mode.

Figure 43:
Important Registers

Register	Address	Parameter
CR_UART	0x0C3	**UART_DATA_MSG_LEN**: Defines the length of data which can be included in interrupt message **UART_DATA_MSG_ADR**: Defines the start address of included data for messaging. **UART_IRQ_CLR_MODE**: Defines the mode how the interrupt state of GP30 is cleared **UART_HB_MODE**: Enables the high baud rate for interrupt messaging **UART_HBR**: Selects the high baud rate for interrupt messaging.

5.2.2 CRC Handling in TDC-GP30

The communication protocol of the UART includes a CRC generation which can be handled by the default settings of GP30, or by settings which can be configured by customer.

However, an initial UART communication must always start with the default CRC settings.

Figure 44:
Important Registers

Register	Address	Parameter
CR_UART	0x0C3	**UART_CRC_MODE**: Selects between default or configured settings if operating in flow meter mode. **UART_CRC_INIT_VAL**: Defines configurable init value for CRC **UART_CRC_ORDER**: Defines configurable order of CRC **UART_CRC_POLY**: Defines configurable CRC polynomial
SHR_RC	0x0DE	**UART_CRC_MODE**: Selects between default or configured settings for initial communication or if operating in flow meter mode.

5.2.3 CRC Handling in Remote Software

CRC has to be generated in reversed order, because UART byte transmission is performed in reversed order with LSB first.

Polynomial: 0x8408 (inverted polynomial of 0x1021)

Initial: 0xFFFF

CRC code generation can be done in low performance algorithm with bit handling or high performance table driven algorithm with byte handling.

Pipelining of

- Transmitting/receiving bytes by UART &
- CRC code updating by CPU core

helps that CRC generation does not minimize performance of UART transmission.

Low Performance Algorithm

This algorithm can be used directly.

```
function crc16reverse(Buffer:String;Polynom,Initial:Cardinal):Cardinal;

var

  i,j                   : Integer;

begin

Result:=Initial;

for i:=1 to Length(Buffer) do begin

  Result:=Result xor ord(buffer[i]);

  for j:=0 to 7 do begin

    if (Result and $0001)<>0 then Result:=(Result shr 1) xor Polynom

    else Result:=Result shr 1;

    end;

  end;

end;
```

Example code (written in Delphi Pascal)

High Performance Algorithm

This table driven algorithm need to initialize a table first (pre-generated with program code or generated on the fly). This can be done by the bit routine (Crc16Reverse in examples).

The table will always be 256 word large with a word size of 16 bit.

Example code (written in Delphi Pascal):

```
var

  CrcTable              : Array[0..255] of Cardinal;

function GenerateTableCrc16Reverse(Poly:Cardinal):Cardinal;

var

  i                     : Cardinal;

begin

for i:=0 to 255 do CrcTable[i]:=Crc16Reverse(chr(i),Poly,0);

end;

function Crc16ByteReverse(Buffer:String;Initial:Cardinal):Cardinal;

var

  i                     : Cardinal;

begin

Result:=Initial;

for i:=1 to Length(Buffer) do begin

  Result:=(Result shr 8) xor CrcTable[(ord(Buffer[i]) xor Result) and $ff];

  end;

end;
```

CRC Unreversing

For beta release, GP30 expects and generates CRC in unreversed order. This needs an unreversing of the generated CRC, but causes additional program code in remote software.

For final release GP30 will expect & generate CRC in reversed order, so this function can be omitted.

Example code (written in C):

```
unsigned int crc

crc = ((crc >> 1) & 0x5555) | ((crc & 0x5555) << 1) // swap odd and even bits

crc = ((crc >> 2) & 0x3333) | ((crc & 0x3333) << 2) // swap consecutive pairs

crc = ((crc >> 4) & 0x0F0F) | ((crc & 0x0F0F) << 4) // swap nibbles

crc = ((crc >> 8) & 0x00FF) | ((crc & 0x00FF) << 8) // swap bytes
```

5.3 Pulse Interface

TDC-GP30 has an integrated pulse interface module which can be used for a simple output of flow volume portions. This pulse interface provides information in the same way a mechanical flow counter would do. It is thus possible to build a stand-alone flow meter with TDC-GP30, which can directly replace mechanical flow counters.

The features of the pulse interface are described in detail in the subsequent sections. In overview, the pulse interface has the following features:

- The pulse interface can provide flow volume information. It outputs one signal pulse for a given flow volume, for example 1 per 1/10 l.
- The pulse interface is highly configurable as described in section 5.3.3. For convenience it is recommended to use build-in ROM routines, as described in section 5.3.2.
- The maximum number of pulses per second is about 100.
- Several GPIOs can be configured as pulse outputs, see GP30_Vol1, section 7.3.3.
- The pulse interface can operate in two modes, pulse and direction or positive and negative pulse, using two GPIOs (see GP30_Vol1 section 6.1.1). With only positive flow one GPIO is sufficient.
- The pulse interface is a separate and independent module of the chip. It operates in principle independent of the CPU. But of course it typically receives its input from a running firmware.
- The pulse interface itself does not perform any flow calculation, it just transforms input values into a corresponding number of output pulses, according to its configuration.
- The pulse interface can also signal error, see section 5.3.2
- Note that the pulse interface can consume considerable current when loaded with a low impedance, since it can operate with high output signal (at Vcc) over long time portions. It is therefore recommended to connect the pulse interface to high impedance inputs in the MΩ region only.

5.3.1 Basic Function

The pulse interface basically receives as input a number of pulses using register **SHR_NI_NPULSE**, and outputs them according to its timing configuration. Note that **SHR_NI_NPULSE** is not necessarily an integer number, it is a signed fractional number in fd24 format. The pulse interface hardware

accumulates values written here at each update and erases the input, such that the next value can be written before the next update. The frequency of updates is also defined by configuration or by the firmware. For details see section 5.3.3. As soon as the accumulated number exceeds one pulse, the pulse interface will generate one pulse, or more at higher input numbers, and subtract the number of the generated pulses from the internally accumulated sum.

The pulse interface itself does not perform complicated calculations, it does not transform an increase of volume into a number of pulses and so on. For this reason, the pulse interface is not easily controlled by its basic configuration parameters. For convenience, the build-in TDC-GP30 ROM library features free ROM routines for simple configuration and operation of the pulse interface, as described in the section 5.3.2.

It should be clear that the pulse interface requires cyclic input over **SHR_NI_NPULSE**, which is typically provided by some active firmware. It is thus straightforward to make use of the ROM routines when writing the firmware for GP30.

5.3.2 Usage with ROM Routines and ams Firmware

To simplify the operation of the pulse interface, the build-in TDC-GP30 ROM library features the following ROM routines:

Figure 45:
ROM Routines for Pulse Interface

ROM Routine	Description
ROM_CFG_PULSE_IF	This routine configures the pulse interface with the parameters calculated from the given configuration.
ROM_PI_UPD	Pulse Interface Update Routine
ROM_PP_PI_UPD	Pulse Interface Update Routine with input from RAM
ROM_RECFG_PULSEIF_FOR_ERROR1	Reconfiguring the pulse interface outputs GPIO0 and GPIO1 as normal GPIOs to signal an error
ROM_SIGNAL_ERROR_ON_PULSEIF1	Signaling error on the pulse interface GPIO0 and GPIO1
ROM_RECFG_PULSEIF_FOR_PULSE1	Configuring GPIO0 and GPIO1 as pulse interface outputs

ROM_CFG_PULSE_IF is used to configure the pulse interface once at startup. It needs as input the desired pulse valence (the number of pulses per liter) and the maximum flow, as well as information about the actual measurement rate. Please see GP30_vol2 section 5.1.1 for details. This routine sets all necessary configuration values for the pulse interface in **SHR_PI_AU_NMB**, **SHR_PI_AU_TIME**, **SHR_PI_TPA** as well as the **PI_TPW** bits in **CR_PI_E2P**. It also calculates a factor called FLOW_SCALE_FACT which is needed for later operation of the pulse interface. The FLOW_SCALE_FACT calculates the number of pulses from a given flow at the given measurement rate. It is returned by the routine in register X.

After this configuration is done, the pulse interface can be operated by regularly calling ROM_PI_UPD or ROM_PP_PI_UPD to update the pulse output. The difference between these two routines is only that ROM_PP_PI_UPD uses the RAM cell RAM_R_FLOW_LPH directly as flow input, while ROM_PI_UPD requires a flow input over register X. Again please see GP30_vol2 section 5.1.1 and GP30_vol4 for details.

When these ROM routines are used, the pulse interface will generate one pulse for each portion of the flow volume as configured. It will operate with the widest possible pulse and as uniformly as possible - it will not produce double pulses, as long as the given maximum flow is not exceeded. Note that the number of pulses per liter is chosen by the user – it should be selected such that at maximum flow not more than 100 pulses per second are required. For example, at a maximum flow of 3600l/h and 10 pulses per liter, only 10 pulses per second would be generated at maximum flow.

The **ams** firmware uses these routines as described and needs only the two firmware data cells FWD_R_PULSE_PER_LITER and FWD_R_PULSE_MAX_FLOW for configuration.

The three remaining ROM routines ROM_RECFG_PULSEIF_FOR_ERROR1, ROM_SIGNAL_ERROR_ON_PULSEIF1 and ROM_RECFG_PULSEIF_FOR_PULSE1 can be used for simple error signaling over the pulse interface. Since these routines are fixed for pulse interface output over GPIO0 and GPIO1, they are not used by the **ams** firmware – **ams** firmware is more flexible and lets the user freely chose the GPIOs for pulse output. Still the way of signaling errors remains the same: In case of error, the chosen pulse output goes permanently high while the direction output toggles. In a custom firmware, the user can take his own decision when to switch to error signal by calling ROM_RECFG_PULSEIF_FOR_ERROR1 and ROM_SIGNAL_ERROR_ON_PULSEIF1. To return to normal operation, ROM_RECFG_PULSEIF_FOR_PULSE1 must be called. The **ams** firmware is configurable and can react to various error conditions, see GP30_Vol4 chapter 2 and sections 2.6 and 5.5.

In error case, the pulse output is permanently raised to signal the error. Of course, this generates an additional pulse and thus a slightly wrong volume count – which may be acceptable since anyway an error happened. The ams firmware is setup to toggle the direction signal each time an error condition is raised, such that this wrong count does not accumulate, but cancels out when it happens frequently.

5.3.3 Configuration Parameters

As discussed above, the integrated pulse interface hardware does not perform any volume calculations. It just knows about the (accumulated) number of pulses to generate, and about some configurations when, at which rate and with which width it should generate them.

In the following the configuration parameters of the pulse interface are described for completeness. These parameters are also available over the GP30 PC software. It is, however, not recommended to use these parameters directly, it is more safe and also easier to use ROM routines as described in the preceding section.

The configuration parameters are part of the configuration register CR_PI_E2P or system handling variables. The input variable for cyclic updates is SHR_PI_NPULSE. All relevant variables are summarized in the following, with some explanations. Note that the ROM routines described in the section define and overwrite many of these parameters.

Figure 46:
CR_PI_E2P Register

Addr: 0x0C1		CR_PI_E2P (Interfaces Control)		
Bit	**Bit Name**	**Default**	**Format**	**Bit Description**
7:0	**PI_TPW**	1	UINT [7:0]	Pulse Interface, Pulse Width = **PI_TPW** * 976.5625 µs (**LP_MODE** = 1), = **PI_TPW** * 1 ms (**LP_MODE** = 0)
8	**PI_EN**	0	Bit	Pulse Interface Enable, if operating in flow meter mode 0: Pulse Interface disabled 1: Pulse Interface enabled
9	**PI_OUT_MODE**	0	Bit	0: Output of pulses on 1 line with additional direction signal 1: Output of pulses on different lines for each direction
10	**PI_UPD_MODE**	0	BIT	0: Automatic Update disabled, only by **PI_UPD** in **SHR_EXC** 1: Automatic Update wit next TOF Trigger This bit defines the frequency of general updates, at which the pulse interface reads and clears **SHR_PI_NPULSE**

Figure 47:
SHR_PI_NPULSE Register

Addr: 0x0D4		SHR_PI_NPULSE (Pulse Interface Number of Pulses)		
Bit	**Bit Name**	**Default**	**Format**	**Bit Description**
31:0	**PI_NPULSE**	0	SINT [31:0]	Number of pulses 1 LSB: $1/2^{24}$ This register is erased at each pulse interface update

Figure 48:
SHR_PI_TPA Register

Addr: 0x0D5		SHR_PI_TPA (Pulse Interface Time Pulse Distance)		
Bit	**Bit Name**	**Default**	**Format**	**Bit Description**
15:0	**PI_TPA**	0	UINT [15:0]	Minimal distance between two pulses 1 LSB: 0.97656 ms (**LP_MODE** = 1) 1 LSB: 1 ms (**LP_MODE** = 0) Mandatory condition: **PI_TPA** > **PI_TPW**
31:16	**NOT USED**	0		Not used

Figure 49:
SHR_PI_IU_TIME Register

Addr: 0x0D6		SHR_PI_IU_TIME (Pulse Interface Internal Update Time)		
Bit	**Bit Name**	**Default**	**Format**	**Bit Description**
15:0	**PI_IU_TIME**	0	UINT [15:0]	Time between two internal updates 1 LSB: 0.97656 ms (**LP_MODE** = 1) 1 LSB: 1 ms (**LP_MODE** = 0) Mandatory condition: **PI_IU_TIME** > 2 and **PI_IU_TIME** > **PI_TPW**
31:16	**NOT USED**	0		Not used

Figure 50:
SHR_PI_IU_NO Register

Addr: 0x0D7		SHR_PI_IU_NO (Pulse Interface Number of Auto Updates)		
Bit	**Bit Name**	**Default**	**Format**	**Bit Description**
7:0	**PI_IU_NO**	0	UINT [7:0]	Number of internal updates between two general updates Recommended condition for uniformed pulse generation: (**PI_IU_NO** + 1) * **PI_IU_TIME** = **TOF_RATE** * **MR_CT**
31:8	**NOT USED**	0		Not used

6 Miscellaneous Functions

6.1 Watchdog

After a system reset the watchdog of TDC-GP30 is enabled. The nominal value of the watchdog time is 13.2 seconds, based on the internal oscillator clock source of 10 kHz.

For operation in time conversion mode, it could be useful to disable the watchdog of TDC-GP30. For that a disable code has to be written to register CR_WD_DIS.

When TDC-GP30 operates with firmware, it is good practice to keep the watchdog enabled. The watchdog timer must then be reset before the watchdog time elapsed. Otherwise the watchdog issues a system reset. Typically the watchdog timer is reset in each post processing cycle by the firmware command clrwdt. The firmware can then use the watchdog for any safety mechanism where a system reset is required to resolve problems, just by not resetting the watchdog timer. And of course the system reset will be triggered when the firmware does not run at all. Of course it must be made sure that the chip starts operating as desired after a system reset, for example by a suitable configuration and setting the bootloader release code in FWD.

Note that after system reset all volatile data is lost, so this is a solution only for serious error cases. It is also good practice to check if a system reset happened, for example by using some free RAM cell as flag after startup – when it returned to zero or to an arbitrary value, probably a reset had happened. Another method is to check the time stamp, see next section.

Figure 51:
Important Register

Register	Address	Parameter
CR_WD_DIS	0x0C0	**WD_DIS**: 0x48DB_A399: Watchdog disabled all other: Watchdog enabled

6.2 TIMESTAMP

TDC-GP30 has a simple timestamp function with a resolution of 1 s. The time stamp counter can be cleared and the current value of the time stamp counter can be latched into readable register.

After system reset, the time stamp is also cleared. Thus the time stamp can be used to check when the most recent system reset happened.

Figure 52:
Important Registers

Register	Address	Parameter
CR_CPM	0x0C5	**TSV_UPD_MODE**: Timestamp Update Mode: set to “1” for automatic update
SHR_XC	0x0DD	**TSV_UPD**: Updates Timestamp
SRR_TS_HOUR	0x0E6	**TS_HOUR**: Accumulated hours of time stamp counter
SRR_TS_MIN_SEC	0x0E7	**TS_MIN**: Accumulated minutes of time stamp counter **TS_SEC**: Accumulated seconds of time stamp counter

6.3 Backup Handling

Backup handling in GP30 can realized by connecting an external I²C EEPROM to the GPIO Unit of the GP30.

Backup handling is triggered by the integrated general purpose timer, which defines the cycle time for the backup. It is enabled if CPU_REQ_EN_GP is assigned to GP Timer.

The backup data has to be written by firmware code “General Purpose Handling” to the well configured EEPROM interface, where backup data is directly transferred to an external I²C EEPROM.

For details on backup handling or different usage of the EEPROM interface please contact support.

Figure 53:
EEPROM Interface

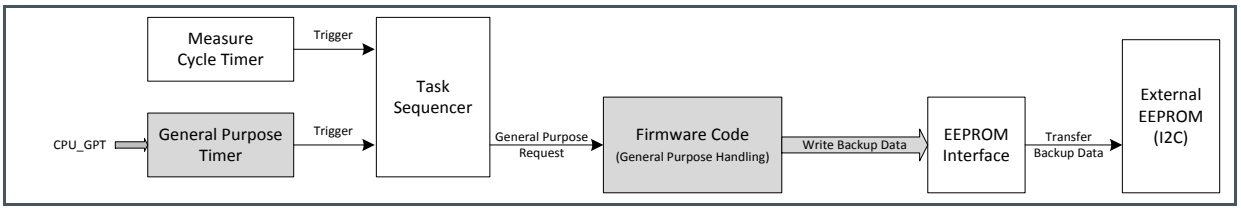


Figure 54:
Important Registers

Register	Address	Parameter
CR_PI_E2P	0x0C1	**E2P_MODE**: Defines which GPIOs are used for I^2C communication **E2P_ADR**: Address for external I^2C EEPROM **E2P_PU_EN**: Enables internal pull-ups for I^2C lines
CR_IEH	0x0C4	**CPU_REQ_EN_GPH**: CPU Request Enable General Purpose **CPU_GPT**: General Purpose Timer
CR_MRG_TS	0x0C6	**GPH_MODE**: General Purpose Handling Mode (has to be set to 1)

6.4 Error Handling

TDC-GP30 features a number of hardware flags to indicate measurement errors. Some of the possible errors are related to wrong configurations, others to measurement problems. Typical error cases are TOF timeouts, when for example a watr flow meter pipe ran try and no signal was received, or a task sequencer timeout when the configured cycle time did not leave enough time to finish all tasks (see section 1.6). Other examples would be short cuts or lost connections on temperature sensors. Manual GP30_Vol4, chapter 5 discusses error handling in detail, and also describes measures taken in **ams** firmware, which go beyond simple hardware errors and feature for example bubble detection.

It is in general recommended to check error flags, with or without firmware usage, and to implement processes to resolve the indicated problems.

Figure 21 sketches the relations between error flag enabling and signaling as implemented in TDC-GP30 hardware. It is possible to generate an interrupt after an error (set IRQ_EN_ERR_FLAG in CR_IEH), and it is possible to configure which errors should be ignored (bits 15..0 in CR_IEH). The **ams** firmware adds several error counters and higher order error conditions to control measurement validity. Please see manual GP30_Vol4, chapter 5 for details.

Figure 55:
Error Handling

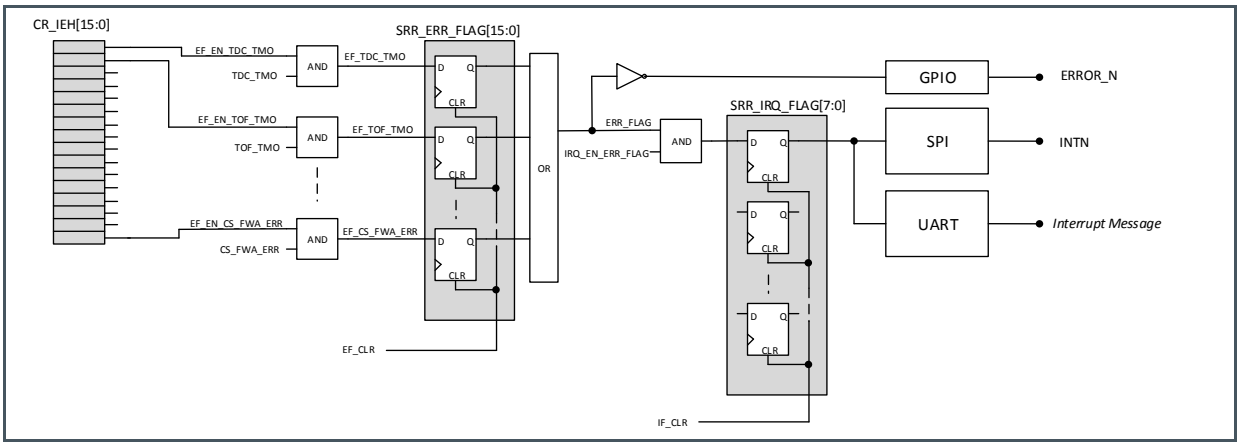


Figure 56:
Important Registers

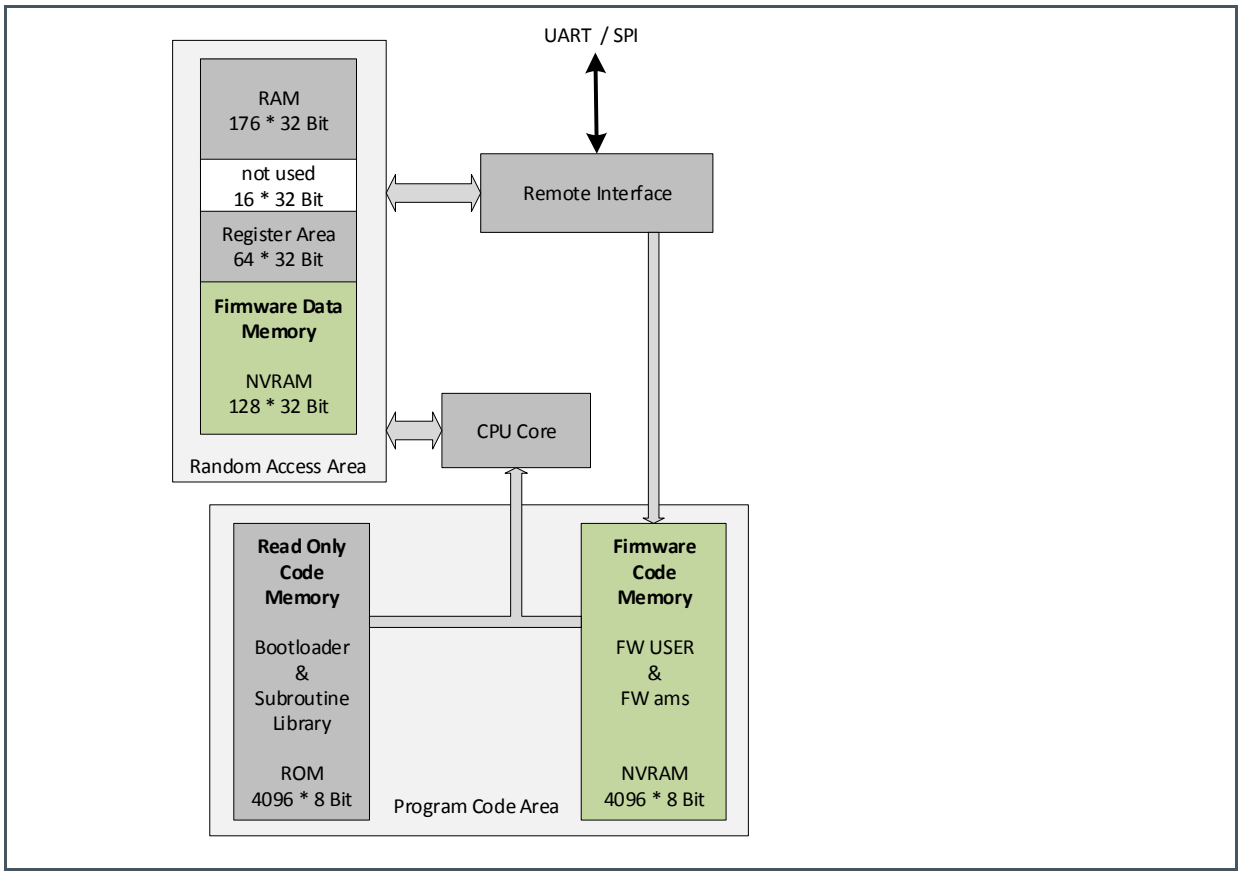
Register	Address	Parameter
CR_IEH	0x0C4	**EF_EN_XX_XX_XX** (Bit 15..0): Enables Error Flags corresponding to register SRR_ERR_FLAG **IRQ_EN_ERR_FLAG**: Interrupt Request Enable for Error Flag
SHR_EXC	0x0DD	**IF_CLR**: Clears Interrupt Flag Register **EF_CLR**: Clears Error Flag Register
SRR_IRQ_FLAG	0x0E0	**ERR_FLAG**: Error Flag has been set
SRR_ERR_FLAG	0x0E1	**EF_XX_XX_XX** (Bit 15..0): Error flags corresponding to error flag enable bits in CR_IEH

7 Handling Firmware with TDC-GP30

7.1 Firmware Location

Due to the Harvard architecture of the GP30 CPU, the programmable firmware in GP30 is located in two non-volatile memories (NVRAMs), one for the firmware program code (FWC) and one for the firmware data (FWD, e.g. parameters, configuration data).

Figure 57:
Firmware Location



The Firmware Code Memory (FWC) is part of the Program Code Area and has a write-only access via the remote interface for programming.

The Firmware Data Memory (FWD) is part of the Random Access Area, where all data memories are located and a read-write-access via the remote interface is given.

Figure 58:
FW Code Memory, Size: 4096 * 8 Bit

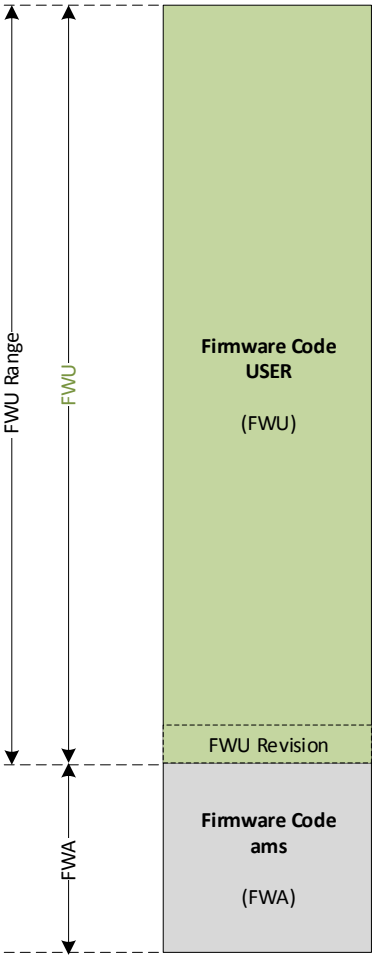
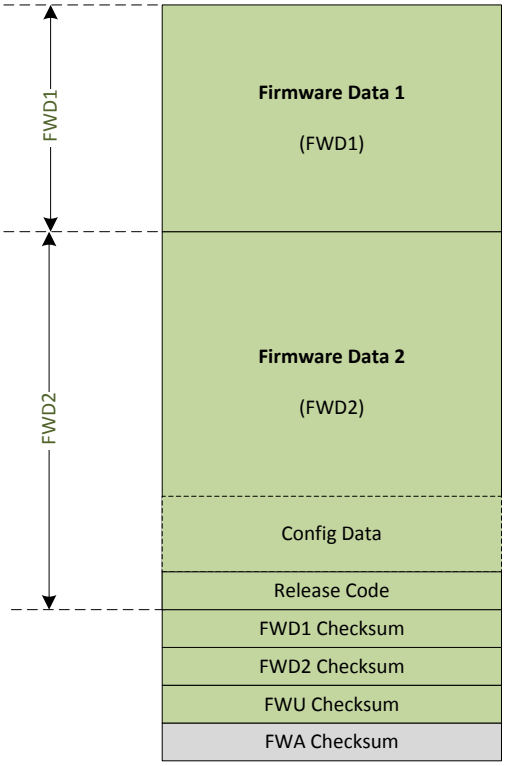


Figure 59:
Firmware Data Memory, Size: 128 * 32 Bit



Green areas are intended for USER programming.

Figure 60:
Important Registers

Register	Address	Parameter
SRR_FWU_RNG	0x0EC	The length of the firmware code, which is reserved for user programming, can be read out from here
SRR_FWU_REV	0x0ED	The last four bytes of the Firmware Code USER is reserved for implementing a revision code by the user. After programming the user firmware code and a system reset of the GP30, the revision of the firmware user code can be read from here
SRR_FWA_REV	0x0EE	The revision of the FW Code ams can be read out from here

7.2 NVRAM Architecture

Each of the two NVRAMs in GP30 is a combination of a volatile SRAM and a non-volatile FLASH memory. Access to/from NVRAM is only given via the SRAM part, where volatile data can be read and written in an unlimited number of times, while non-volatile data resides in FLASH part.

Read and write accesses can be addressed in any order.

The complete data transfer from SRAM to FLASH is performed by a STORE. From FLASH to SRAM the data are completely transferred by a RECALL. The execution of both transactions has to be enabled first.

When storing user data and code, FW_STORE_LOCK can be used. Then firmware data cannot be read back, and the data and code content of the chip can only be erased, not modified (see section 7.5). When such a locked chip is erased, it unlocks again. The protected **ams** code part in such a chip will not be erased and remains unchanged.

Figure 61:
NVRAM

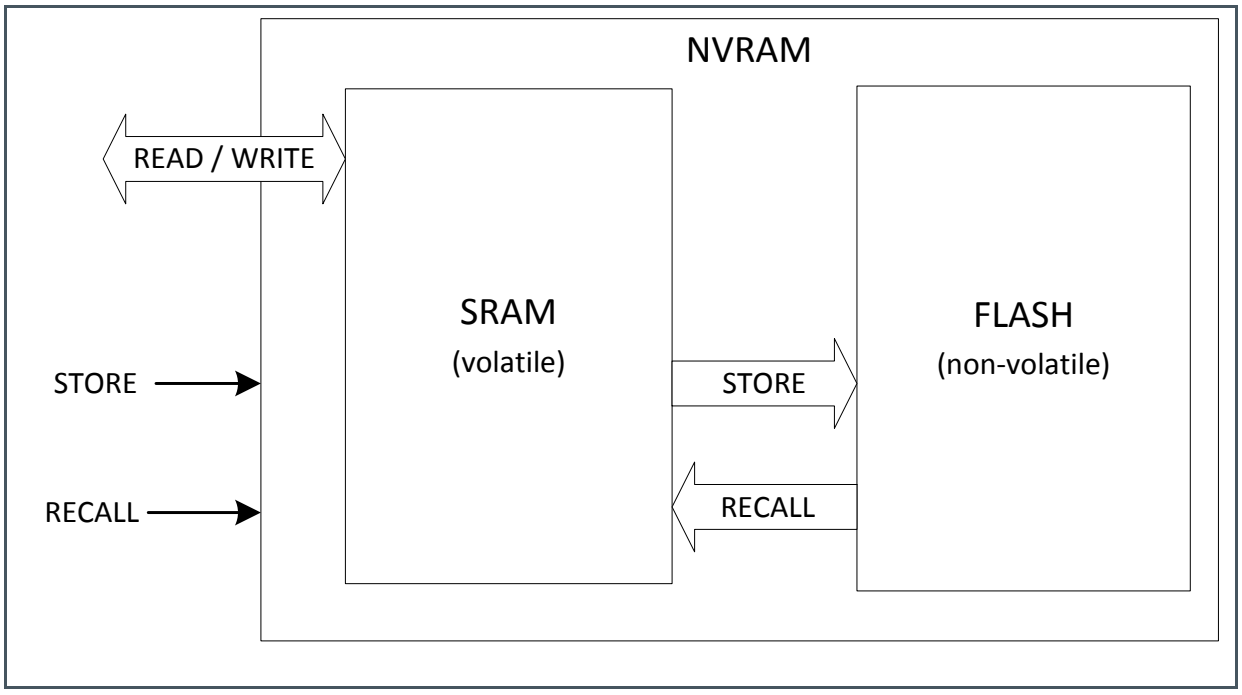


Figure 62:
Important Registers

Register	Address	Parameter
SHR_FW_TRANS_EN	0x0DF	Enables FW Transactions STORE or RECALL executed in **SHR_RC**

Register	Address	Parameter
SRR_RC	0x0DE	**FW_STORE**: Stores Firmware (Code & Data) **FW_STORE_LOCK**: Stores & Lock Firmware (Code & Data) **FW_ERASE**: Erases user part Firmware Code & Firmware Data **FWC_RECALL**: Recalls Firmware Code **FWD_RECALL**: Recalls Firmware Data
SRR_IRQ_FLAG	0x0E0	**FW_TRANS_FNS**: Firmware Transaction Finished (only if **IM_FW_TRANS_FNS** is set in **CR_IEH**)

7.3 Download Firmware to NVRAMs

The download of Firmware is defined as a write of FW Code and/or FW Data into NVRAMs of GP30 with a followed store to the non-volatile part of the NVRAM.

Figure 63:
Important Registers

Register	Address	Parameter
BLD_RLS	0x17B	Release Code for bootloader 0xABCD_7654: Bootloader execution is released any other: Bootloader not released After system reset a proper configuration of GP30 is only executed if bootloader release code is programmed
FWD1_CS_EXP	0x17C	FW Data 1 Checksum Expected
FWD2_CS_EXP	0x17D	FW Data 2 Checksum Expected
FWU_CS_EXP	0x17E	FW Code USER Checksum Expected
FWA_CS_EXP	0x17F	The FW ams Code and its expected checksum is pre-programmed by **ams** and need not to be programmed by user.

The following sections define the recommended sequences via remote interface for SPI.

The preparation, storing and completion sequences are always the same for all 3 download options.

The defined opcodes & data are in principal the same for UART, but the different protocol characteristics for UART communication needs to be considered (MSB/LSB sequence, Little Endian, block length byte, CRC, acknowledge).

7.3.1 Download FW Code & Data

Figure 64:
Download FW Code & Data

Step	Description	Opcodes & Data	Sequence
1	Execute System Reset by sending **RC_SYS_RST**	0x99	Preparation
2	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 0x48DBA399	
3	Execute Measure Cycle Off by sending **RC_MCT_OFF**	0x8A	
4	Set **HS_CLK_SEL** dependent on used high speed clock (Typical: 4 MHz) by writing code to **SHR_RC**	0x5A 0xDE *) 0x00000004	
5	Enable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x50F5B8CA	
6	Write Firmware User Code to GP30 by separating complete code in block transfer of max. 128 data bytes.	0x5C 0x00 0x00 0xXX 0xXX ….. 0x5C 0x00 0x80 0xXX 0xXX ….. …….	Writing
7	Write Firmware Data to FWD addresses 0-126 including **BLD_RLS**, **FWD1_CS_EXP**, **FWD2_CS_EXP, FWU_CS_EXP**	0x5B 0x00 (1) 0xXXXXXXXX 0xXXXXXXXX …….	
8	Clear interrupt flag register by sending RC_IF_CLR	0x8D	Storing
9	Execute **FW_STORE** / **FW_STORE_LOCK** (dependent on LOCK condition) by writing code to SHR_RC	0x5A 0xDE 0x00010000 or 0x00020000	
10	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
11	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion
12	Disable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x00000000	

(1) Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x5A for writing in RAM or register area, 0x5B for writing to FWD in NVRAM)

7.3.2 Download FW Code Only

Figure 65:
Download FW Code Only

Step	Description	Opcodes & Data	Sequence
1	Execute System Reset by sending **RC_SYS_RST**	0x99	Preparation
2	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 (1) 0x48DBA399	
3	Execute Measure Cycle Off by sending **RC_MCT_OFF**	0x8A	
4	Set **HS_CLK_SEL** dependent on used high speed clock (Typical: 4 MHz) by writing code to **SHR_RC**	0x5A 0xDE 0x00000004	
5	Enable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x50F5B8CA	
6	Write Firmware User Code to GP30 by separating complete code in blocks transfer of max. 128 data bytes.	0x5C 0x00 0x00 0xXX 0xXX ….. 0x5C 0x00 0x80 0xXX 0xXX ….. …….	Writing
7	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Recalling FW Data
8	Execute FWD_RECALL by writing code to **SHR_RC**	0x5A 0xDE 0x00100000	
9	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
10	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Storing
11	Execute **FW_STORE** / **FW_STORE_LOCK** (dependent on LOCK condition) by writing code to **SHR_RC**	0x5A 0xDE 0x00010000 / 0x00020000	
12	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
13	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion
14	Disable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x00000000	

(1) Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x5A for writing in RAM or register area, 0x5B for writing to FWD in NVRAM)

7.3.3 Download FW Data Only

Figure 66:
Download FW Data Only

Step	Description	Opcodes & Data	Sequence
1	Execute System Reset by sending **RC_SYS_RST**	0x99	Preparation
2	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 (1) 0x48DBA399	
3	Execute Measure Cycle Off by sending **RC_MCT_OFF**	0x8A	
4	Set **HS_CLK_SEL** dependent on used high speed clock (Typical: 4 MHz) by writing code to **SHR_RC**	0x5A 0xDE 0x00000004	
5	Enable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x50F5B8CA	
6	Write Firmware Data to FWD addresses 0-126 including **BLD_RLS**, **FWD1_CS_EXP**, **FWD2_CS_EXP**, **FWU_CS_EXP**	0x5B 0x00 (1) 0xXXXXXXXX 0xXXXXXXXX …….	Writing
7	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Recalling FW Code
8	Execute **FWC_RECALL** by writing code to **SHR_RC**	0x5A 0xDE 0x00080000	
9	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
10	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Storing
11	Execute **FW_STORE** / **FW_STORE_LOCK** (dependent on LOCK condition) by writing code to **SHR_RC**	0x5A 0xDE 0x00010000 / 0x00020000	
12	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
13	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion
14	Disable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x00000000	

(1) Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x5A for writing in RAM or register area, 0x5B for writing to FWD in NVRAM)

7.4 Verify Methods of Stored Data

To verify that FW code & FW data is correctly stored, the GP30 provides two methods:

- Checksum Generation
- Comparing Memory Content

Both methods utilize the volatile content of NVRAM's SRAM part. To make sure that non-volatile content is checked a RECALL has to be executed first. Please remember that a RECALL can be performed separately for the Firmware Code Memory and the Firmware Data Memory.

After a system reset or a system init, a RECALL is automatically executed for both memories.

7.4.1 Checksum Generation

Checksum generation is performed for 4 different memory parts:

- Firmware Data 1 (FWD1)
- Firmware Data 2 (FWD2)
- Firmware Code USER (FWU)
- Firmware Code ams (FWA)

The checksum generation can be executed as part of the bootloading sequence, as well by invoking remote command **RC_FW_CHKSUM**.

After checksum generation the checksums are temporary placed at following RAM addresses:

Figure 67:
Checksums in RAM

Address	Checksum
0x0A8	FWD1_CS
0x0A9	FWD2_CS
0x0AA	FWU_CS
0x0AB	FWA_CS

The checksum generation routine also compares the generated checksums to the expected checksums, programmed in FWD memory:

Figure 68:
Checksums in FWD

Address	Checksum
0x17C	FWD1_CS_EXP
0x17D	FWD2_CS_EXP

Address	Checksum
0x17E	FWU_CS_EXP
0x0AB	FWA_CS

Finally the checksum generation routine sets appropriate status flags in register SHR_GPO. These status flags are directly connected to error flags in register SRR_ERR_FLAG and can be integrated in error handling.

The checksums are built by simply adding all bytes of the memory parts:

- Firmware Code USER: Each address provides 1 byte to be added to checksum.
- Firmware Data 1, 2: Each address provides 4 single bytes to be added to checksum.

Recommended sequence for verifying by checksum generation:

Figure 69:
Checksum Verification

Step	Description	Opcodes & Data	Sequence
1	Execute System Reset by sending **RC_SYS_RST**	0x99	Preparation
2	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 (1) 0x48DBA399	
3	Execute Measure Cycle Off by sending **RC_MCT_OFF**	0x8A	
4	Set **HS_CLK_SEL** dependent on used high speed clock (Typical: 4 MHz) by writing code to **SHR_RC**	0x5A 0xDE 0x00000004	
5	Enable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x50F5B8CA	
6	Read **FW_UNLOCKED** from **SRR_MSC_STF**	0x7A 0xEA *read data (Bit 2)*	Getting FW -Lock State -Range -Revisions
7	Read **FWU_RNG** from **SRR_FWU_RNG**	0x7A 0xEC *read data*	
8	Write FWU code for “GET_REV” at address 0 “GET_REV” is a small firmware routine which allows the revision read out of FW User Code & FW ams Code	0x5C 0x00 0x00 0xCA 0xF0 0x6B 0xF2 0xDC 0x0B 0xCD	
9	Execute **GET_REV** by writing code to **SHR_CPU_REQ**	0x5A 0xDC 0x00000008	

Step	Description	Opcodes & Data	Sequence
10	Read **SRR_FWU_REV** & **SRR_FWA_REV**	0x7A 0xED *read data* *read data*	
11	Write 0x00 to FWC addresses 0-4095 by separating complete code in blocks transfer of max. 128 data bytes.	0x5C 0x00 0x00 0x00 0x00 ….. 0x5C 0x00 0x80 0x00 0x00 ….. …….	Clearing FW Code & FW Data
12	Write 0x00000000 to FWD addresses 0-127	0x5B 0x00 (1) 0x00000000 0x00000000 …….	
13	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Recalling FW Code
14	Execute **FWC_RECALL** by writing code to **SHR_RC**	0x5A 0xDE 0x00080000	
15	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
16	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Recalling FW Data
17	Execute **FWD_RECALL** by writing code to **SHR_RC**	0x5A 0xDE 0x00100000	
18	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
19	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Checksum Generation & Verify
20	Execute checksum generation by sending **RC_FW_CKSUM**	0xB8	
21	Check on interrupt **CHKSUM_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 3)*	
22	Read generated & expected checksum for FWD1 and compare	0x7A 0xA8 (1) *gen. checksum* 0x7B 0x7C (1) *exp. checksum*	
23	Read generated & expected checksum for FWD2 and compare	0x7A 0xA9 *gen. checksum* 0x7B 0x7D *exp. checksum*	
24	Read generated & expected checksum for FWU and compare	0x7A 0xAA *gen. checksum*	

Step	Description	Opcodes & Data	Sequence
		0x7B 0x7E *exp. checksum*	
25	Read generated & expected checksum for FWA and compare	0x7A 0xAB *gen. checksum* 0x7B 0x7F *exp. checksum*	
26	Clear interrupt flag register by sending RC_IF_CLR	0x8D	Completion
27	Disable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x00000000	

(1) Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x5A for writing in RAM or register area, 0x5B for writing to FWD in NVRAM)

7.4.2 Comparing Memory Content

Another method is to compare memory content directly with expected data. This can be done by invoking a remote request with command RC_FWC_CMP for FW Code Memory or with command RC_RAA_CMP for FW Data Memory. The data block to be compared needs a minimum of 16 consecutive addresses.

Please contact **ams** for further support.

7.5 Firmware Lock & Erase

The GP30 has the ability of a firmware lock, which causes:

- A write protection for Firmware Code
- A read protection for Firmware Data
- A read protection for configuration registers

Please keep in mind, that Firmware Code is read protected all the time.

A firmware lock is performed by following steps:

- Make sure that desired Firmware Code & Firmware Data is written to SRAM part of NVRAMs
- Enable FW Transactions in SHR_FW_TRANS_EN.
- Perform LOCK by executing FW_STORE_LOCK in SHR_RC
- The end of the LOCK transaction is signalized by interrupt (SPI) or interrupt message (UART)

To unlock firmware, GP30 provides an ERASE, which erases Firmware User Code and Firmware Data.

A firmware erase is performed by following steps:

- Enable FW Transactions in SHR_FW_ TRANS_EN.
- Perform LOCK by executing FW_STORE_LOCK in SHR_RC
- The end of the LOCK transaction is signalized by interrupt (SPI) or interrupt message (UART)

Figure 70:
Important Registers

Register	Address	Parameter
SHR_FW_TRANS_EN	0x0DF	Enables FW Transactions STORE or RECALL executed in **SHR_RC**
SRR_RC	0x0DE	**FW_STORE_LOCK**: Stores and locks Firmware (Code & Data) **FWC_ERASE**: Erases Firmware
SRR_IRQ_FLAG	0x0E0	**FW_ TRANS _FNS**: Firmware Transaction Finished (only if **IM_FW_ TRANS _FNS** is set in **CR_IEH**)

Recommended sequence for erase with included deadlock break

Figure 71:
Firmware Erase

Step	Description	Opcodes & Data	Sequence
1	Execute Bus Master Request by sending **RC_BM_REQ**	0x88	Deadlock Break
2	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 0x48DBA399 (1)	
3	Execute Bus Master Request by sending **RC_BM_REQ**	0x88	
4	Disable Watchdog by writing code to **CR_WD_DIS**	0x5A 0xC0 0x48DBA399	
5	Execute Measure Cycle Off by sending **RC_MCT_OFF**	0x8A	
6	Execute SV initialization by sending **RC_SV_INIT**	0x9C	
7	Execute FEP initialization by sending **RC_FEP_INIT**	0x9D	
10	Enable **IRQ_EN_FW_TRANS_FNS** in **CR_IEH**	0x5A 0xC4 0xXXX2XXXX	Preparation

Step	Description	Opcodes & Data	Sequence
11	Set **HS_CLK_SEL** dependent on used high speed clock (Typical: 4 MHz) by writing code to **SHR_RC**	0x5A 0xDE 0x00000004	
12	Enable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x50F5B8CA	
13	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Erasing FW
14	Execute **FW_ERASE** by writing code to **SHR_RC**	0x5A 0xDE 0x00040000	
15	Check on interrupt **FW_TRANS_FNS** (FW transaction finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 1)*	
16	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion
17	Disable FW Transaction by writing code to **SHR_FW_TRANS_EN**	0x5A 0xDF 0x00000000	

(1) Please keep in mind that the most significant bit of RAA address is coded as the least significant bit of opcode in the byte before (e.g. 0x5A for writing in RAM or register area, 0x5B for writing to FWD in NVRAM)

8 Appendix

8.1 Notational Conventions

Throughout the GP30 documentation, the following stile formats are used to support efficient reading and understanding of the documents:

- Hexadecimal numbers are denoted by a leading 0x, e.g. 0xAF = 175 as decimal number. Decimal numbers are given as usual.
- Binary numbers are denoted by a leading 0b, e.g. 0b1101 = 13. The length of a binary number can be given in bit (b) or Byte (B), and the four bytes of a 32b word are denoted B0, B1, B2 and B3 where B0 is the lowest and B3 the highest byte.
- Abbreviations and expressions which have a special or uncommon meaning within the context of GP30 application are listed and shortly explained in the list of abbreviations, see section 8.2. They are written in plain text. Whenever the meaning of an abbreviation or expression is unclear, please refer to the glossary at the end of this document.
- **Variable names** for hard coded registers and flags are in bold. Meaning and location of these variables is explained in the datasheet (see registers CR, SRR and SHR).
- Variable names which represent memory or code addresses are in grey. Many of these addresses have a fixed value inside the ROM code, others may be freely defined by software. Their meaning is explained in the firmware and ROM code description, and their physical addresses can be found in the header files. These variable names are defined by the header files and thus known to the assembler as soon as the header files are included in the assembler source code. Note that different variable names may have the same address, especially temporary variables.
- *Physical variables* are in italics (real times, lengths, flows or temperatures).

8.2 Abbrevations

Figure 72:
Abbrevations

Short	Description
AM	Amplitude measurement
CD	Configuration Data
CPU	Central Processing Unit
CR	Configuration Register
CRC	Cyclic Redundancy Check
DIFTOF, DIFTOF_ALL	Difference of up and down → TOF

Short	Description
DR	Debug Register
FEP	Frontend Processing
FDB	Frontend data buffer
FHL	First hit level (physical value V_{FHL})
FW	Firmware, software stored on the chip
FWC	Firmware Code
FWD	Firmware Data
FWD-RAM	Firmware Data memory
GPIO	General purpose input/output
Hit	Stands for a detected wave period
HSO	High speed oscillator
INIT	Initialization process of → CPU or → FEP
IO	Input/output
I2C	Inter-Integrated Circuit bus
LSO	Low speed oscillator
MRG	Measurement Rate Generator
NVRAM, NVM	Programmable Non-Volatile Memory
PI	Pulse interface
PP	Post Processing
PWR	Pulse width ratio
R	RAM address pointer of the CPU, can also stand for the addressed register
RAA	Random Access Area
RAM	Random Access Memory
RI	Remote Interface
ROM	Read Only Memory
ROM code	Hard coded routines in ROM
SHR	System Handling Register
SPI	Serial Peripheral Interface
SRAM	Static RAM
SRR	Status & Result Register
SUMTOF	Sum of up and down TOF
Task	Process, job
TDC	Time-to-digital-converter
TOF, TOF_ALL	Time of Flight

Short	Description
TS	Task Sequencer
TM	Temperature measurement
UART	Universal Asynchronous Receiver & Transmitter
USM	Ultrasonic measurement
V_{ref}	Reference voltage
X,Y,Z	Internal registers of the CPU
ZCD	Zero cross detection, physical level V_{ZCD}

8.3 Glossary

Figure 73:
Glossary

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
AM	Amplitude measurement	This is a peak measurement of the received signal amplitude. It can be configured to be executed in different time frames, which allows to pick the overall signal maximum (to control the signal level), or to measure only the peak of a selected number of → wave periods. The latter allows for a more detailed receive signal analysis.
Backup	Permanent storage of a data copy	TDC-GP30 Volume 3 User Manual is prepared for an external data backup, foreseen over the built-in I^2C-bus, which permits write and read with an external EEPROM. In principle, a user may also utilize the → GPIOs for his own interface implementation for external backup.
Bootloader	System routine that initializes CPU operation	Typically after a system reset, first time when the →TS calls the → CPU, the bootloader routine is called. If the → firmware is released, the bootloader loads the chip configuration from FWD into CR and does other hardware initializations like reading firmware revision numbers and calculation of checksums.
Burst	Analog signal containing a number of → wave periods	For a flow measurement, a → fire burst, that means a fixed number of → wave periods of the measurement frequency, is send over a
Calibration	Parameter adjustment to compensate variations	In TDC-GP30 Volume 3 User Manual, different calibration processes are implemented and needed for high quality measurements: → Firmware calibrations: Flow and temperature calibration, but also the → FHL adjustment are under full control of the firmware. Half-automated calibrations: → AM calibration and → HSO calibration are based on dedicated measurements, initiated by the → TS on demand. The actual calibrations need further evaluation by the firmware. Fully hard-coded calibrations: these calibrations need no interaction from firmware. One example is → ZCD level calibration, which only needs to be initiated by the → TS frequently. Another example is → TDC calibration which happens automatically before each measurement.
CD	Configuration Data	16 x (up to) 32b words of → flash memory for configuration of the chip, address range 0x16C - 0x17A (→ NVRAM). Is copied to → CR for actual usage.
Comparator	Device that compares two input signals	See → ZCD-comparator

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
CPU	Central Processing Unit	32b processor (Harvard architecture type) for general data processing. The CPU has a fixed instruction set and acts directly on its three input- and result-registers → X,Y and Z as well as on addressed RAM. The fourth register of the CPU is the → RAM address pointer R. Instructions for the CPU are read as → FWC or → ROM code at an address given by the → program counter.
CR	Configuration Register	The chip actually uses for its hardware configuration a copy of the → CD into the CR address range 0x0C0 - 0x0CF (see → direct mapped registers).
C0G		Material of a ceramic capacitor with a very low temperature drift of capacity
DIFTOF, DIFTOF_ALL	Difference of up and down → TOF	The difference between up and down → TOF is the actual measure for flow speed. (see also → SUMTOF). DIFTOF_ALL is the DIFTOF using → TOF_ALL results, averaged over all TOF → hits
Direct mapped registers	Registers with direct hardware access	These register cells are not part of some fixed memory block, they rather have individual data access. This makes them suitable for hardware control. See → SHR, → SRR, → CR and → DR. Labels have the according prefix.
FEP	Frontend Processing	Task of the → TS where frontend measurements are performed
FDB	Frontend data buffer	Part of the → RAM where the → frontend temporarily stores its latest measurement results (→ RAA address range from 0x80 up to maximally 0x9B)
FHL, V_{FHL}	First hit level	Voltage level similar to the → ZCD level, but shifted away from Zero level, for save detection of a first → hit. The FHL determines, which of the → wave periods of the receive → burst is detected as first hit. It thus has a strong influence on → TOF and must be well controlled, in order to achieve comparable TOF measurements.
Fire, fire burst, fire buffer	Send signal → burst	The measurement signal on sending side is called fire burst, its output amplifier correspondingly fire buffer.
Firmware	Program code (in a file) for chip operation	The program code can be provided by **ams** or by the customer, or a combination of both. The complete program code becomes the → FWC (firmware code) when stored in the → NVRAM. The term firmware is in general used for all firmware programs, no matter if they make up the complete FWC or not.
Flow meter mode	Operation mode of TDC-GP30 Volume 3 User Manual as full flow meter system	In flow meter mode, the TDC-GP30 Volume 3 User Manual also performs further evaluation of → TOF results, to calculate physical results like flow and temperature. To do this, it uses a → firmware running on its internal CPU. See for comparison → time conversion mode
Frontend	Main measurement circuit block	This part of the TDC-GP30 Volume 3 User Manual chip is the main measurement device, containing the analog measurement interface (including the → TDC). The frontend provides measurement results which are stored in the → FDB.
FWC	Firmware Code	Firmware code denotes the complete content of the → NVRAM's 4kB section (address range 0x0000 to 0x 0FFF). The difference to the term → firmware is on the one hand that firmware means the program in the file. On the other hand, a particular firmware may provide just a part of the complete FWC. FWC is addressed by the CPU's program counter, it is not available for direct read processes like RAM.
FWD	Firmware Data	The firmware configuration and calibration data, to be stored in the → FWD-RAM
FWD-RAM	Firmware Data memory	128 x 32b words of → NVRAM (built as volatile → SRAM and non-volatile flash memory). The FWD-RAM is organized in two address ranges, FWD1 (→ RAM addresses 0x100 - 0x11F) and FWD2 (RAM addresses 0x120 – 0x17F). Main purpose is calibration and configuration Due to its structure, FWD-RAM can be used like usual → RAM by the firmware. But note that with every data recall from flash memory the contents of the SRAM cells get overwritten.

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
GPIO	General purpose input/output	TDC-GP30 Volume 3 User Manual has up to 7 GPIO pins which can be configured by the user. Some of them can be configured as → PI or → I^2C-interface.
Hit	Stands for a detected wave period	The receive → burst is typically a signal which starts with → wave periods of the measurement frequency at increasing signal levels. While the first of these wave periods are too close to noise for a reliable detection, later signal wave periods with high level can be detected safely by the → ZCD-comparator. The comparator converts the analog input signal into a digital signal, which is a sequence of hits. To detect the first hit at an increased signal level, away from noise, the input signal is compared to the → FHL. After the first hit, the level for comparison is immediately reduced to the → ZCD level, such that all later hits are detected at zero crossing (note that the ZCD level is defined to zero with respect to the receive signal, it is actually close to → Vref or another user-defined level). Different hits are denoted according to their usage: □ Hit (in general) stands for any detected → wave period. □ First hit is actually the first hit in a → TOF measurement (not the first wave period!) □ TOF hits means all hits which are evaluated for → TOF measurements. Note that typically the first hit is not a TOF hit. □ Start hit is the first TOF hit. This is typically not the first hit, but (according to configuration) some well-defined later hit. Minimum the 3rd hit has to set as Start hit. □ Stop hit is the last TOF hit. It is also defined by configuration and should not be too close to the end of the receive → burst. □ Ignored hits are all hits which are not evaluated for the TOF measurement: All hits between first hit and start hit, as well any hit between TOF hits or after the stop hit.
HSO	High speed oscillator	The 4 or 8 MHz oscillator of the TDC-GP30 Volume 3 User Manual. In usual operation only switched on when needed, to reduce energy consumption. This is the time base for → TDC measurements. The HSO is typically less accurate that the → LSO. It should be frequently → calibrated against the LSO to obtain the desired absolute accuracy of the → TDC.
INIT	Initialization process of → CPU or → FEP	In TDC-GP30 Volume 3 User Manual terminology, INIT processes do not reset registers or digital IOs, while → reset does at least one of it. Several different INIT processes are implemented, see chapter “Reset hierarchy” for details.
IO	Input/output	Connections to the outside world for input or output
I2C	Inter-integrated circuit bus	Standard serial bus for communication with external chips.
LSO	Low speed oscillator	The 32768 Hz crystal oscillator of the TDC-GP30 Volume 3 User Manual. This oscillator controls the main timing functions (→ MRG and → TS, real time clock).
MRG	Measurement rate generator	The measurement rate generator controls the cyclic → tasks of TDC-GP30 Volume 3 User Manual by setting task requests in a rate defined by configuration (→ CR). When the MRG is activated, it periodically triggers the → TS for initiating the actual → tasks.
NVRAM, NVM	Programmable Non-Volatile Memory	TDC-GP30 Volume 3 User Manual contains two sections of programmable non-volatile memory: One section of 4kB → FWC memory, and another of → FWD-RAM (FWD1:→ RAM addresses 0x100 - 0x11F and FWD2: RAM addresses 0x120 – 0x17F), in total 128 x 32b words. It is organized as a volatile SRAM part which is directly accessed from outside, and a non-volatile flash memory part.
PI	Pulse interface	Standard 2-wire interface for flow output of a water meter. Typically outputs one pulse per some fixed water volume (e.g. one pulse per 0.1 l), while the other wire signals the flow direction. Permits stand-alone operation and is fully compatible to mechanical water meters.
PP	Post Processing	Processing activities of the → CPU, typically after frontend processing (e.g. a measurement) , initiated by –>TS

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
Program counter	Pointer to the current code address of the → CPU	The program counter addresses the currently evaluated → FWC or → ROM-code cell during → CPU operation The program counter always starts at 0xF000, when any CPU action is requested. If any kind of firmware code execution is requested, the program counter is continued at 0x0000 (for FW initialization, post processing or general purpose handling).
PWR	Pulse width ratio	Width of the pulse following the first → hit, related to the pulse width at the start hit. This width indicates the position of the → FHL relative to the level of the detected → wave period and thus gives some information on detection safety (small value means FHL is close to the peak amplitude and the desired wave period may be missed due to noise; large value indicates the danger that an earlier wave period may reach FHL level and trigger the first hit before the desired wave period).
R	RAM address pointer of the CPU	The → CPU acts on the data of the → X-,Y- and Z-register and on one single RAM cell. The pointer R defines the address of the current RAM cell.
RAA	Random Access Area	Address range from 0x000 to 0x1FF covering the → RAM addresses. Memory cells within this address range can all be read, most of them can also be written (except → SRR and → DR). The RAA covers memory cells of different technology: → RAM (including → FDB), → FWD-RAM (including → CD), → direct mapped registers (→ SHR, → SRR, → CR and → DR).
RAM	Random Access Memory	176 x 32b words of volatile memory, used by → FDB and → Firmware. Address range 0x000 to 0x0AF
RAM address	Address of a cell in the RAA range	A RAM address is used by the firmware or over → RI to point to a memory cell for data storage or retrieval. Note that RAM addresses cover not only actual RAM, but all cells in the RAA range. Address range from 0x000 to 0x1FF
Register	Memory cell for dedicated data storage	Memory cells are typically called register when they contain flags or configuration bits, or when they have a single dedicated purpose (see → CPU, → CR, → SHR and → SRR).
Reset	Reset of the chip	TDC-GP30 Volume 3 User Manual has different processes and commands that can call resets and initializations at different levels. Some of them refresh → CR or GPIO state, others just (re-) initialize CPU or frontend. The latter are rather denoted → INIT. See chapter “Reset hierarchy” for details.
RI	Remote Interface	Interface for communication with a remote controller (see → SPI)
ROM	Read Only Memory	4kB of fixed memory, contains hard coded routines for general purpose and parts of **ams**’ → firmware (ROM code). Address range 0xF000 – 0xFFFF. The ROM code is addressed by the CPU’s program counter, it is not available for direct read processes like RAM.
ROM code	Hard coded routines in ROM	See → ROM.
SCL	Serial Clock	Serial clock of I^2C interface
SDA	Serial Data	Serial data of I^2C interface
SHR	Special Handling Register	Registers that directly control chip operation. The data & flags of special handling registers have a dynamic character. They are typically updated by post processing, but have to be initially configured before measurement starts.
SPI	Serial Peripheral Interface	Standard interface for communication of the TDC-GP30 Volume 3 User Manual with an external master controller
SRAM	Static RAM	TDC-GP30 Volume 3 User Manual does not use any dynamic RAM, in fact all RAM in TDC-GP30 Volume 3 User Manual is static RAM. However, the term “SRAM” is in particular used for the RAM-part of the → NVRAM.

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
SRR	Status & Result Register	The SRR-registers describe the current state of the chip. They are set by the chip hardware and contain error and other condition flags, timing information and so on.
SUMTOF, SUMTOF_ALL	Sum of up and down TOF	The sum of up and down → TOF is a measure for the speed of sound in the medium, which can be used for temperature calculation. SUMTOF_ALL is the SUMTOF using → TOF_ALL results, averaged over all TOF → hits.
Supervisor	Functional block of TDC-GP30 Volume 3 User Manual that controls voltage and timing	The supervisor of TDC-GP30 Volume 3 User Manual controls chip operation and timing through the measurement rate generator (→ MRG) and the task sequencer (–>TS). It also covers voltage control and adjustment functions as well as the main oscillators → LSO and >HSO
Task	Process, job	The term task is used for a process which aims at fulfilling some fixed purpose, separate from other tasks with different goals. Typical tasks in TDC-GP30 Volume 3 User Manual are → TOF measurement, temperature measurement (→ TM), post processing (→ PP), remote communication and voltage measurement.
Time conversion mode	Remotely controlled operation of TDC-GP30 Volume 3 User Manual	In time conversion mode, the TDC-GP30 Volume 3 User Manual mainly acts as a → TOF measurement system. It may operate self-controlled or remotely controlled, but it does no further result evaluation. This operation mode is similar to the typical usage of the **ams** chips GP21 and GP22. For comparison see → Flow meter mode
TDC	Time-to-digital-converter	The core measurement device of TDC-GP30 Volume 3 User Manual. Measures times between a start- and a stop-signal at high accuracy and high resolution. The internal fast time base of the TDC is automatically → calibrated against the → HSO before each measurement.
TOF, TOF_ALL	Time of Flight	Basic measurement result for an ultrasonic flow meter: The time between send and receive → burst (with some offset, depending on → hit detection). Measurements of TOF are done in flow direction (down TOF) and in the opposite direction (up TOF). TDC-GP30 Volume 3 User Manual also provides the sum of all TOF → hits in the values TOF_ALL.
TS	Task Sequencer	The task sequencer arranges and initiates the → tasks which are requested by the → MRG in one measurement cycle or which are initiated remotely.
TM	Temperature measurement	This task means a temperature measurement using sensors, in contrast to temperatures which are calculated results from a TOF measurement (see → SUMTOF)
Transducer	Electromechanical conversion device	Transducers for flow measurements are piezoelectric devices that convert an electrical signal into ultrasound and reverse. They are usually matched to the flow medium (e.g. water). TDC-GP30 Volume 3 User Manual can connect directly to the send and receive transducer.
USM	Ultrasonic measurement	The principle of an ultrasonic flow meter is to measure → TOFs of ultrasound in flow direction and against it, and to calculate the flow from the result. See also → transducer.
V_{ref}	Reference voltage	The analog interface of TDC-GP30 Volume 3 User Manual refers to V_{ref}, a nominal voltage for → VZCD of typically 0.7V. This makes it possible to receive a DC-free AC-signal with a single supply voltage. Up to the level of V_{ref}, negative swings of the receive signal are avoided.
V_{ZCD}	Zero cross detection level	This voltage level represents the virtual zero line for the receive → burst. It is normally close to → Vref, just differing by the offset of the → ZCD-comparator. Needs frequent → calibration to compensate the slowly changing offset. Optionally, this voltage can be configured differently in SHR_ZCD… through the firmware.

Term	Meaning	TDC-GP30 Volume 3 User Manual Interpretation
Watchdog, watchdog clear	Reset timer for chip re-initialization	The watchdog of TDC-GP30 Volume 3 User Manual → resets the chip (including → CR refresh) if no watchdog clear (→ firmware command clrwdt) within 13.2s (typically) is executed. This is a safety function to interrupt hang-up situations. It can be disabled for remote control, when no firmware clears the watchdog automatically.
Wave period	One period of the signal wave	A period of typically 1us length for a 1 MHz measurement frequency. This may be a digital pulse, for example when sending, or a more sinusoidal wave when receiving. Fire or receive → bursts are sequences of wave periods.
X-, Y- and Z-register	Input- and result registers of the CPU	The → CPU acts on these → registers for data input and result output.
ZCD	Zero cross detection	All → hits following the first hit are detected when the received signal crosses a voltage level VZCD, defined as zero with respect to the receive → burst. In contrast, the first hit is detected when the received signal crosses the different voltage level VFHL(→ FHL).
ZCD-Comparator	→ comparator for → hit detection	The ZCD-comparator in TDC-GP30 Volume 3 User Manual detects → hits in the received → burst signal by comparing the received signal level to a given reference voltage (see also → FHL, → ZCD and → hit).

9 Revision Information

Changes from previous version DB_GP30_Vol3_171218 to current revision v1-00	Page
Complete transfer to **ams** layout for application notes	All

- Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
- Correction of typographical errors is not explicitly mentioned.

10 Legal Information

Headquarters
ams AG
Tobelbader Strasse 30
8141 Premstaetten
Austria, Europe
Tel: +43 (0) 3136 500 0

Please visit our website at www.ams.com
Buy our products or get free samples online at www.ams.com/Products
Technical Support is available at www.ams.com/Technical-Support
Provide feedback about this document at www.ams.com/Document-Feedback
For sales offices, distributors and representatives go to www.ams.com/Contact
For further information and requests, e-mail us at ams_sales@ams.com