member of the ams group

Ultrasonic-Flow-Converter

Data Sheet

TDC-GP30Y

System-Integrated Solution for Ultrasonic Flow Meters
Volume 3: User Manual

August 13, 2015
Document-No: DB_GP30Y_Vol3_en V0.0

acam-messelectronic gmbh is now a member of ams group

Support / Contact

For direct sales, distributor and sales representative contacts, visit the acam web site at:

www.acam.de www.ams.com

For technical support you can contact the acam support team: support.stutensee@ams.com

or by phone +49-7244-74190.

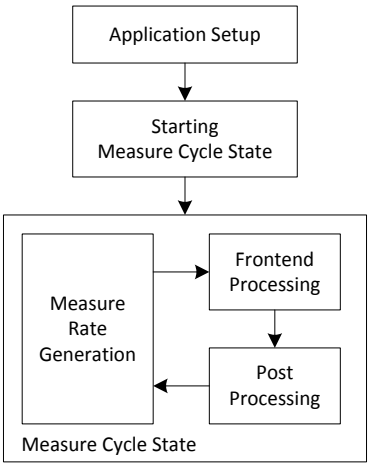
Content

1 Introduction

1.1 Ultrasonic Flow Converter (UFC) Processes

When integrating acam 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: Basis processes



Processes	Description
Application Setup	Initial configuration with application parameters
Measure Rate Generation	Permanently (re-)starting of measurement cycles
Frontend Processing	Controlling the ultrasonic and temperature sensors TDC measurement of ultrasonic and temperature sensors
Post Processing	Conversion of time based frontend results into flow and temperature results

In applications realized with acam's 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:

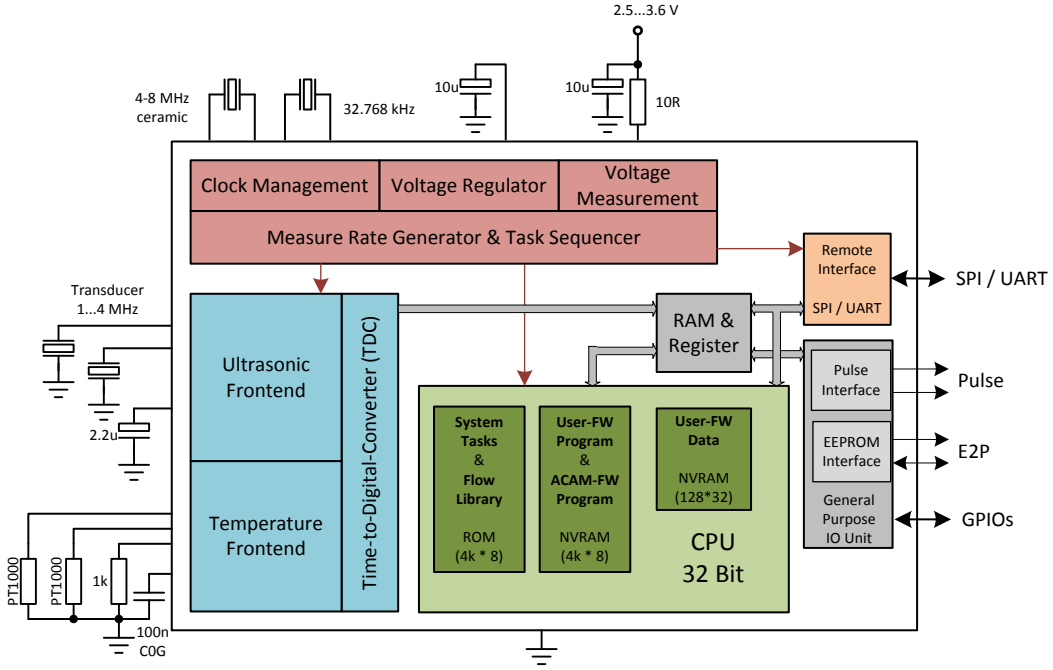
Application Setup	Measure Rate Generation	Post Processing	Operating Mode
by GP30	by GP30	by GP30	Flow meter mode
per Remote		per Remote	Time conversion mode (self-controlled)
	per Remote		Time conversion mode (remote-controlled)
all other combinations			Only for test or debug purpose

Frontend processing is always performed by TDC-GP30.

1.2.1 Flow Meter Mode

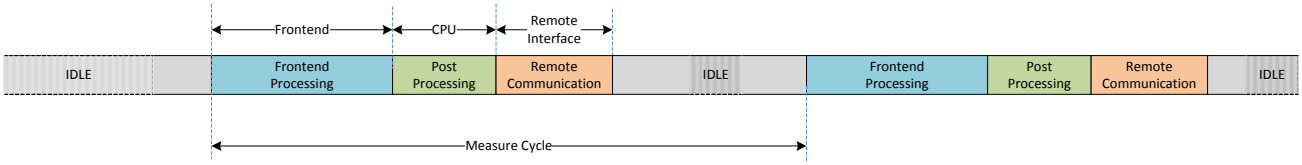
Flow meter mode is the typical application mode for TDC-GP30, where all processes are performed by TDC-GP30. The application setup is performed by a bootloader, executed in the ROM of the integrated CPU. The post processing is realized by a programmed firmware, also executed in the integrated CPU.

Figure 2 Blockdiagram



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

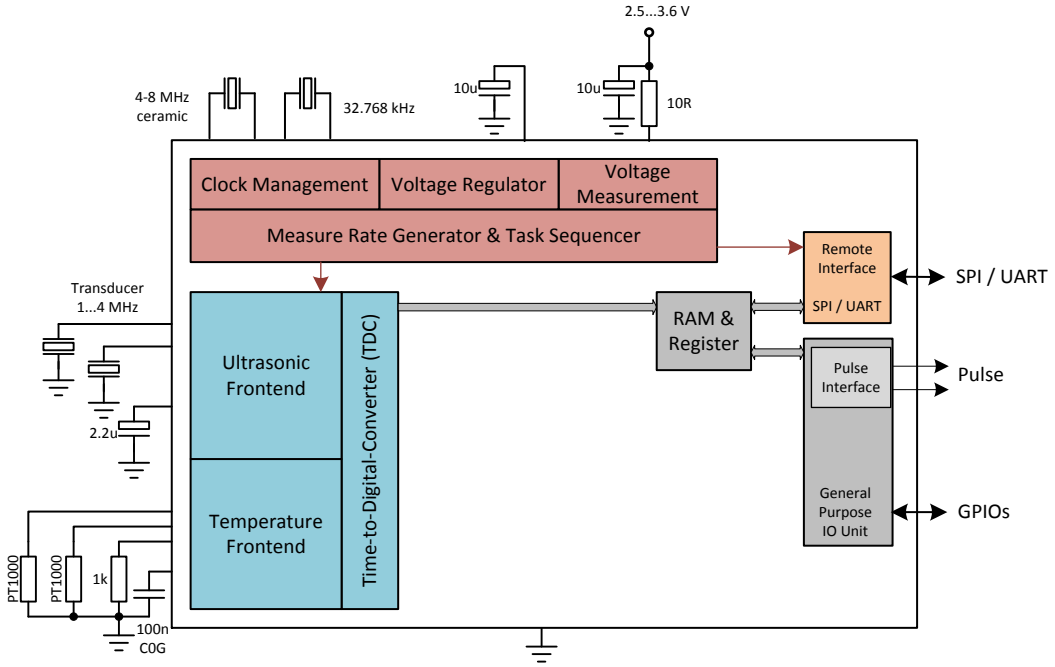
Figure 3 Prcesses



1.2.2 Time Conversion Mode

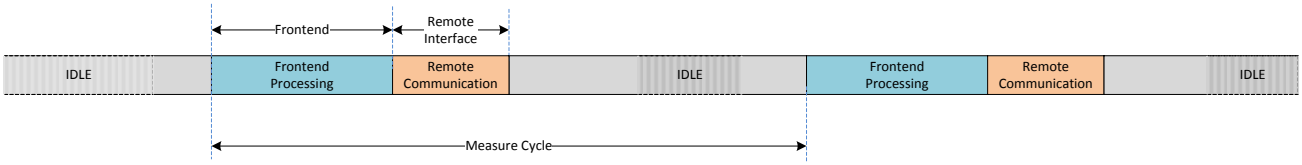
In time conversion mode the application setup and post processing are performed by a remote controller. In case the remote controller also performs the measure rate generation, the GP30 operates in the same mode as GP21/GP22.

Figure 4 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 can't be used, because it's only controllable by programmed firmware in CPU.

Figure 5 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 chapters 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 typical not changed while GP30 is in measure cycle state.

Nevertheless, it’s 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 programmed to the “Configuration Data” section in NVRAM for FW Data (0x16C – 0x17A).
  In case firmware is released after programming the release code to address 0x17B, the configuration data will be transferred by the bootloader from this section 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 typical 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 has to be performed by “FW_INIT”, which is a user programmable FW subroutine, executed once after bootloader has been finished. Typically 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 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 7 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 180°.

Figure 6 Cycle triggers

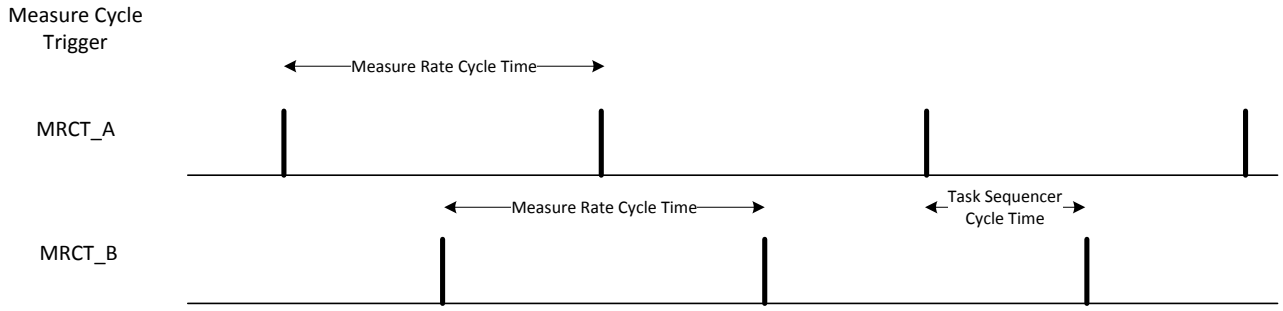
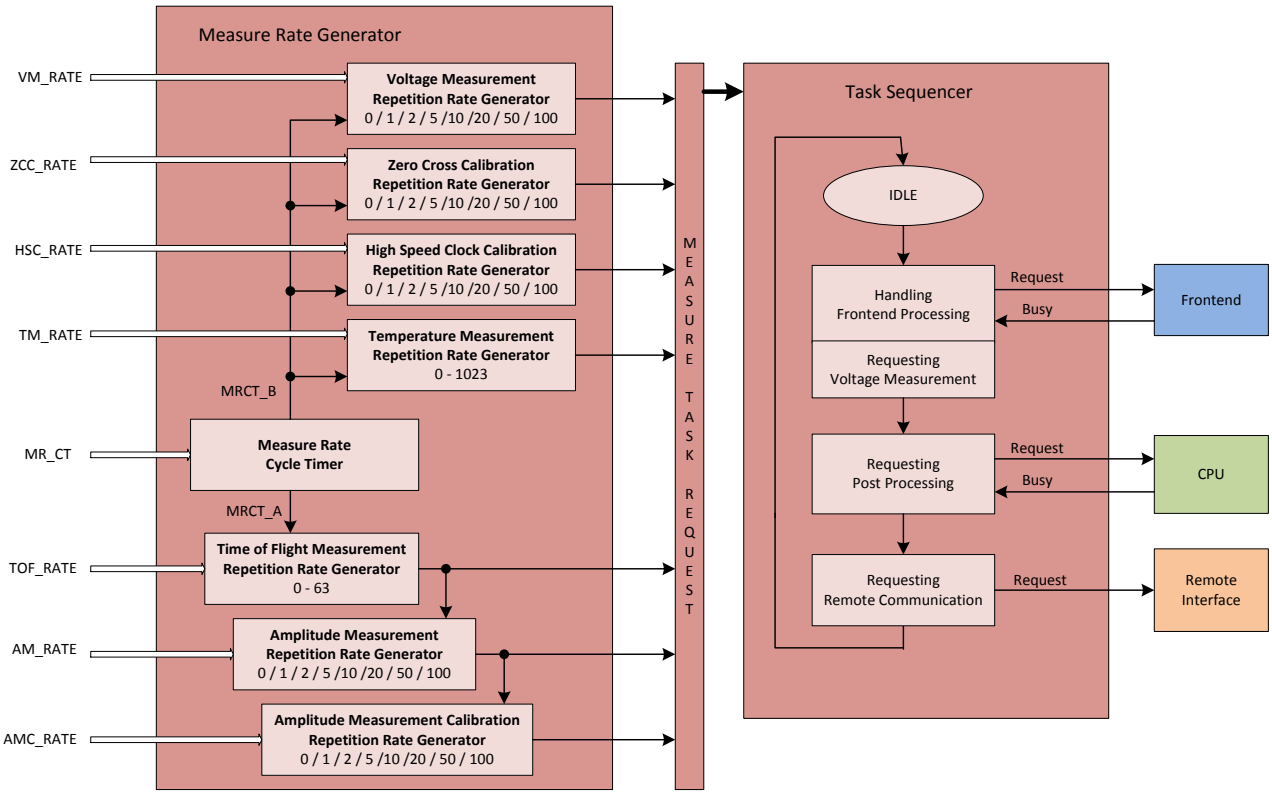


Figure 7 Measure rate generation & task sequencer



The measure rate cycle time can be configured by **MR_CT** in register **CR_MRG_TS**. The time between **MRCT_A** and **MRCT_B** is the resulting time for the task sequencer 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.

Please refer to chapter 7.1, “Calculating Task Sequencer Cycle Time”, for details on how to calculate the task sequencer cycle time.

Important registers:

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**

Typical application

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 sec
TM_RATE	120	Temperature Measurent every 30 sec
HSC_RATE	50	High Speed Clock Calibration every 12,5 sec
ZCC_RATE	20	Zero Cross Calibration every 5 sec
VM_RATE	100	Voltage Measurement every 25 sec

1.6 Clock Management

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

The high speed clock 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.

Important register:

CR_CPM	0x0C5	**HS_CLK_ST**: Defines settling time for high speed clock **HS_CLK_SEL**: Defines the frequency of high speed clock **HSC_RATE**: Defines repetition rate for high speed clock calibration task

1.7 Voltage Measurement

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

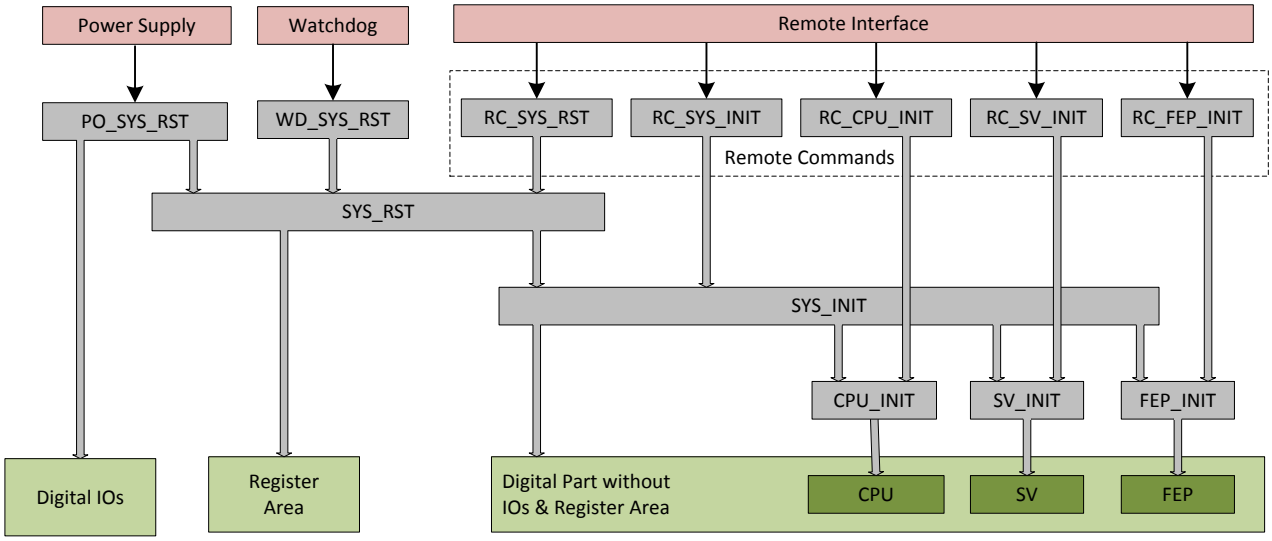
Important registers:

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

1.8 Reset Hierarchy

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

Figure 8 Reset hierarchy



Following resets can be distinguished:

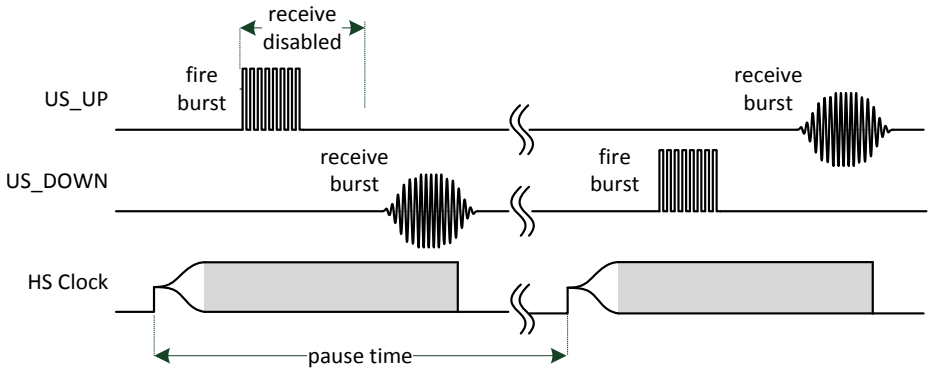
- **PO_SYS_RST**
  Power-On Reset of TDC-GP30, only performed after VDD33 is switched on.
  Resets complete TDC-GP30 including digital IOs.
- **WD_SYS_RST**
  Watchdog System Reset, performed after the watchdog timer expired
  Resets complete digital part of TDC-GP30 without digital IOs
- **RC_SYS_RST**
  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
- **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 SV block of TDC-GP30
- **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 9



A time-of-flight measurement (TOF) starts by first enabling HS clock. After the configured settling time of HS clock has been reached, 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 HS clock 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.

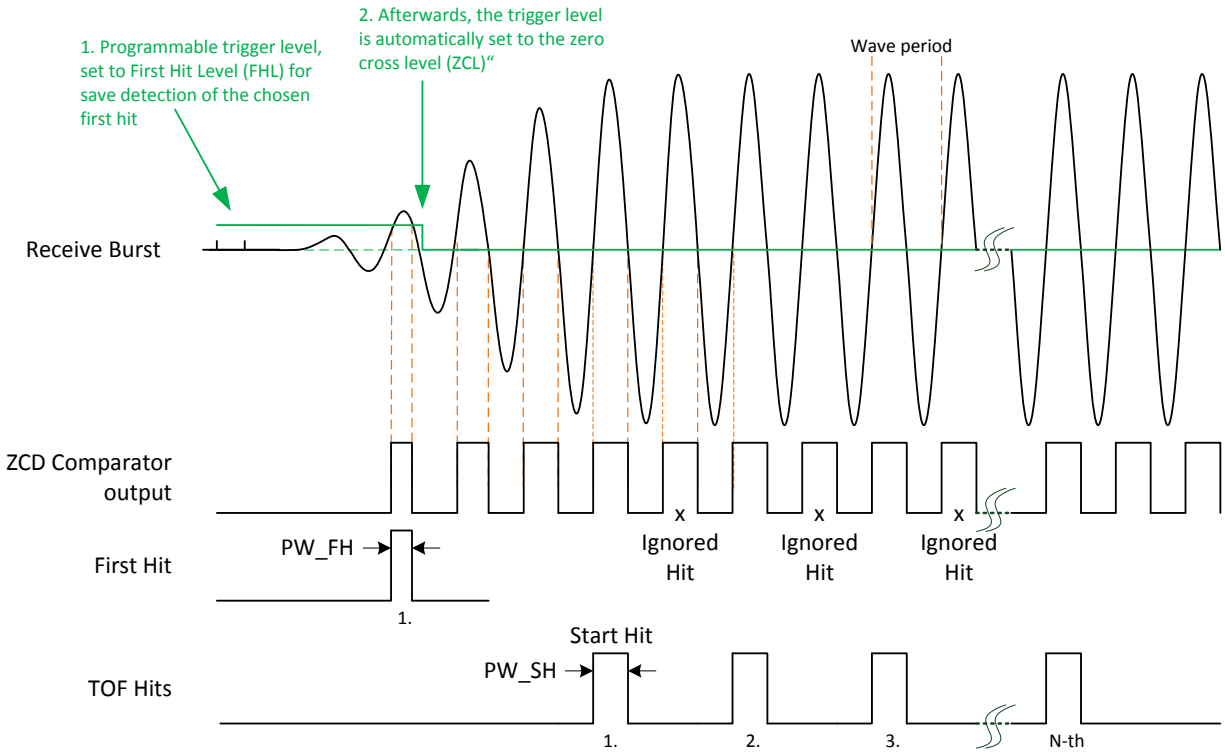
Important registers

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 10



PW_FH = pulse width first hit, PW_SH = pulse width start hit

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 extended 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** and should be in the range of 0.6 to 0.7. The ratio can be used to track the first wave levels for up and for down direction.

Important registers

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 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 can’t be used. In general, wide-bandwidth transducers with fast amplitude growth are preferred.

Important registers

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 is updated automatically if zero cross calibration is enabled to any rate (by setting **ZCC_RATE** > 0). The zero cross level can be read via **SHR_ZCD_LVL**.

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).

Important registers

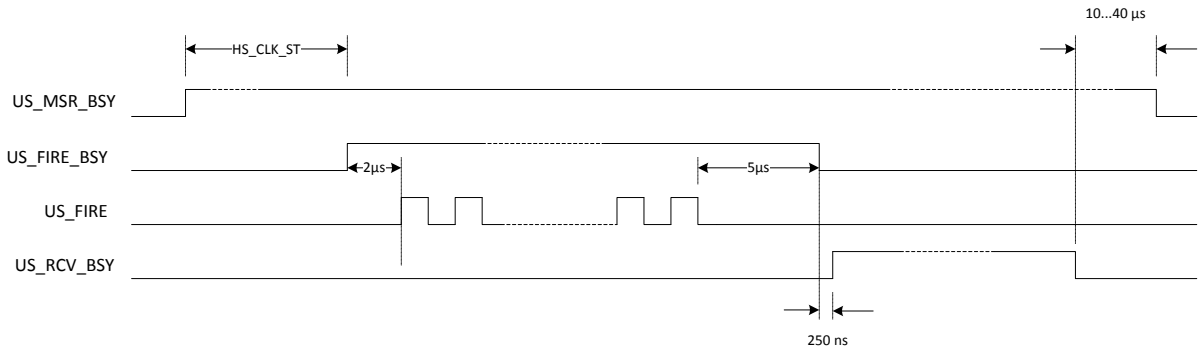
CR_USM_FRC	0x0C9	**ZCC_RATE**: Defines repetition rate for zero cross calibration
SHR_ZCD_LVL	0x0D9	Defines zero cross detection level

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 11



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

Timings are based on a high speed clock of 4 MHz.

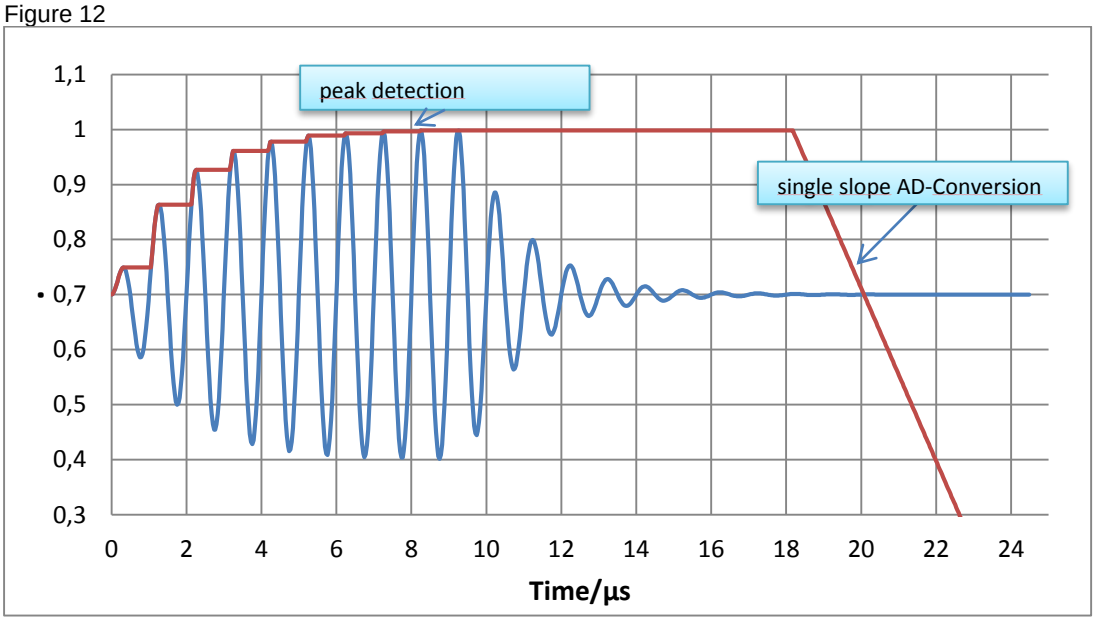
Important register

CR_GP_CTRL	0x0C2	**GP_CTRL**: This register should be configured to 0x0C0C00CC 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 acam 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.

Figure 12



In Time Conversion Mode the result is given as raw time measurement data. Those time data are converted into voltage by means of the following formulas:

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 ≡ $AMC_{high}[ns]$
- FDB_US_AMC_VL ≡ $AMC_{low}[ns]$

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

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

$AMC_{Offset}[mV] = (2 * AMC_L[ns] - AMC_H[ns]) * AMC_{Gradient}\left[\frac{mV}{ns}\right]$

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

AM_PD_END <= **TOF_START_HIT_NO**

Important registers

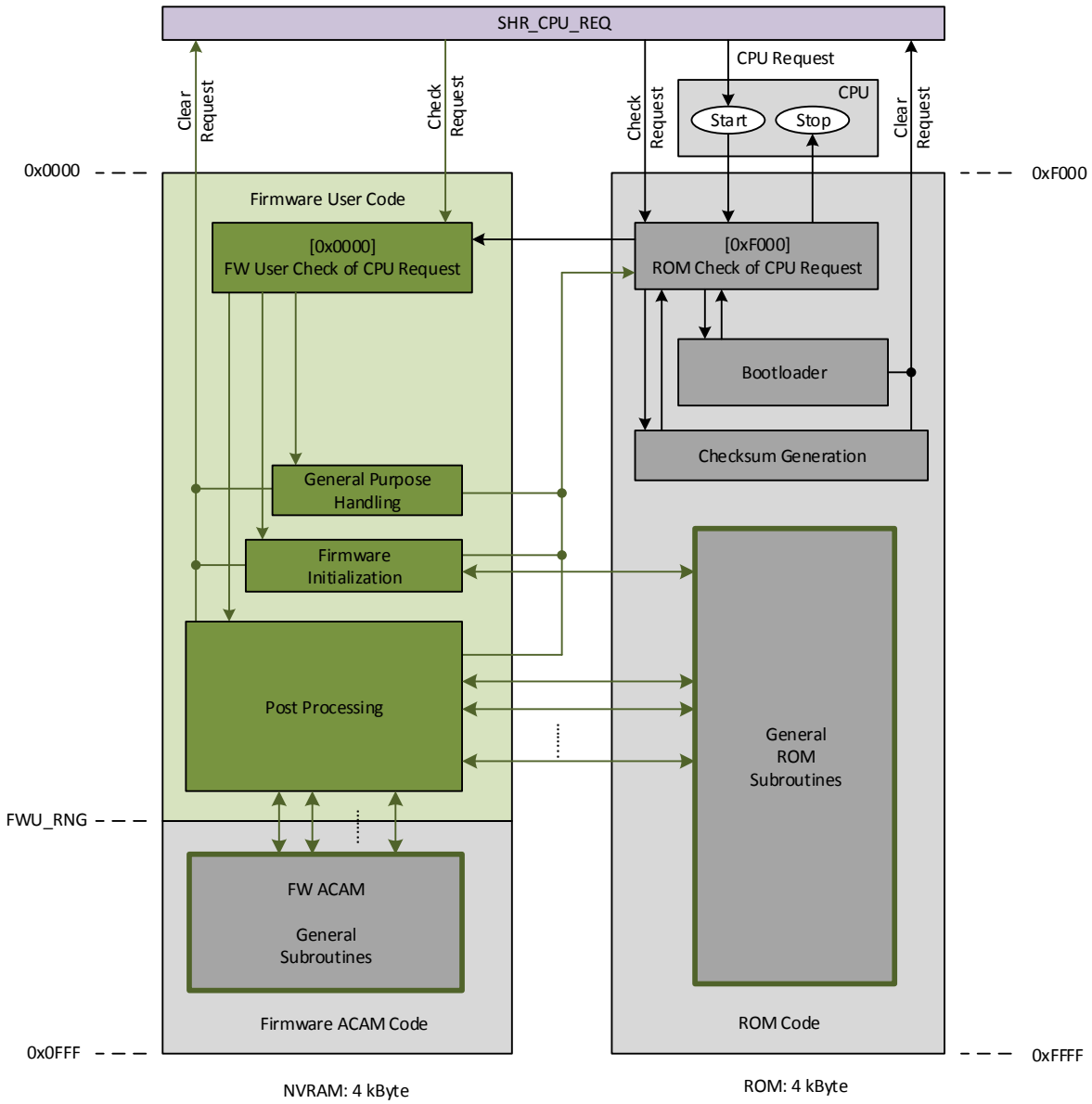
CR_USM_AM	0x0CB	**AM_RATE**: Defines repetition rate for amplitude measurement **AM_MODE**: Defines moment when peak detection starts **AM_PD_END**: Defines number of waves when peak detection stops **AM_PD_WIN**: Defines peak detection window **AMC_RATE**: Defines repetition rate for amplitude measurement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

3 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

Figure 13 CPU Handling



Program code in green color has to be defined and programmed by customers, whereby public subroutines in FW ACAM code or some ROM code can also be used by customer.

3.1 Check of CPU Request

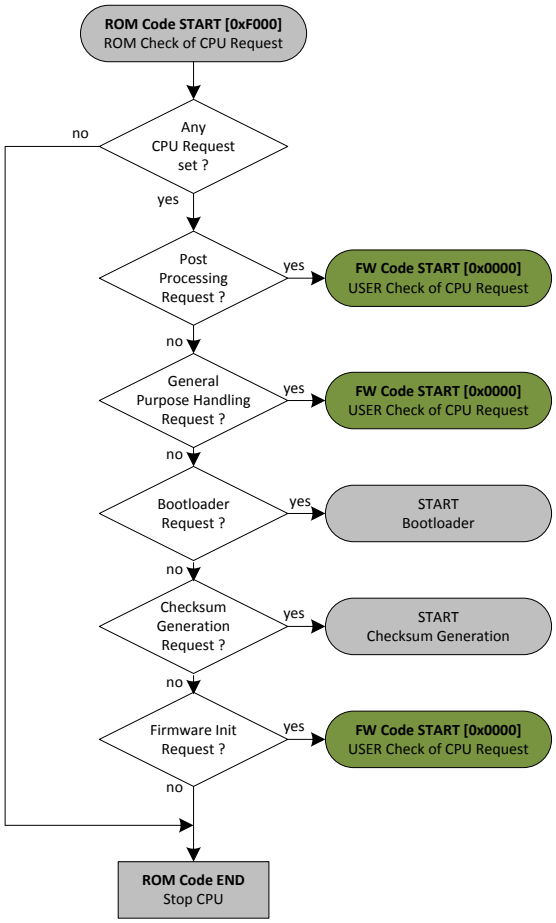
In case any of the requests is set in **SHR_CPU_REQ** the CPU starts first in ROM code with a check of request type.

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

For the other 4 CPU requests, the check has to be repeated 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”.

Figure 14 CPU request handling



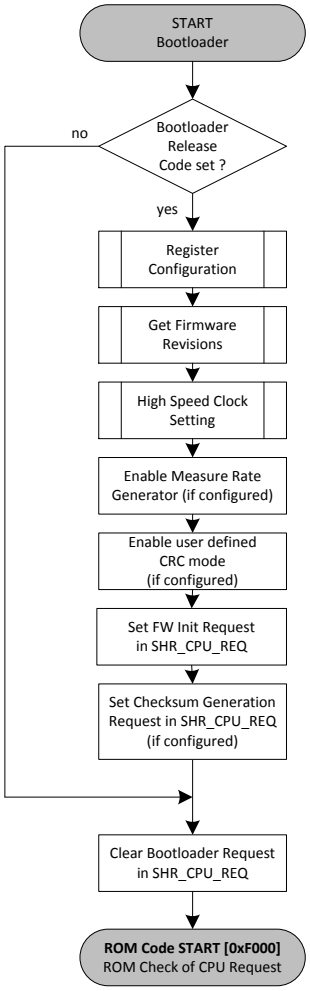
Important registers

SHR_CPU_REQ	0x0DC	CPU Requests

3.2 Bootloader

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

Figure 15



Boatloader 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 has been finished

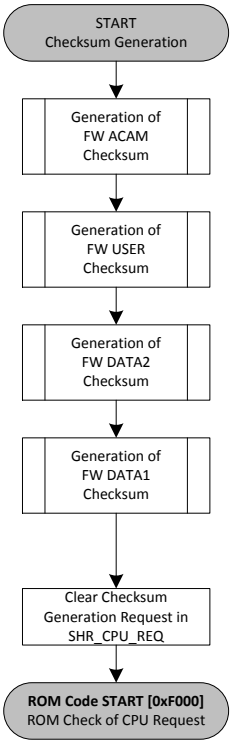
Finally, the bootloader clears its request in **SHR_CPU_REQ** and jumps back to ROM code for checking CPU requests.

Important FW Data

BLD_RLS	0x17B	Release Code for bootloader 0xABCD_7654: Release code for bootload execution all other: Bootloader is not released

3.3 Checksum Generation

Figure 16



Checksum generation can be requested by remote command **RC_FW_CHKSUM** or, if configured, by bootloader.

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.

4 Remote Communication

Remote communication can be started within a task sequencer cycle, after frontend and post processing has been 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 **IM_FW_IRQ_S** in **CR_IEH** has to be enabled. The interrupt can be triggered by firmware by **FW_IRQ_S** in **SHR_EXC** and will be then requested when post processing has been finished. Typically used in flow conversion mode where a remote communication need not be requested with every task sequencer cycle.

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 by setting **FW_IRQ_S** as described above.

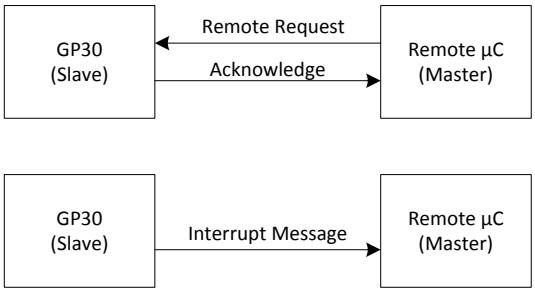
Important registers

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

4.1 UART Communication

4.1.1 Principal Modes

Figure 17



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 to save a followed remote request to get the results.

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.

Important registers

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.

4.1.2 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.

Important registers

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.

4.1.3 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 doesn't 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.

```
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;
```

Example code (written in Delphi Pascal)

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.

```
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
```

Example code (written in C)

4.2 Getting Measurement Results

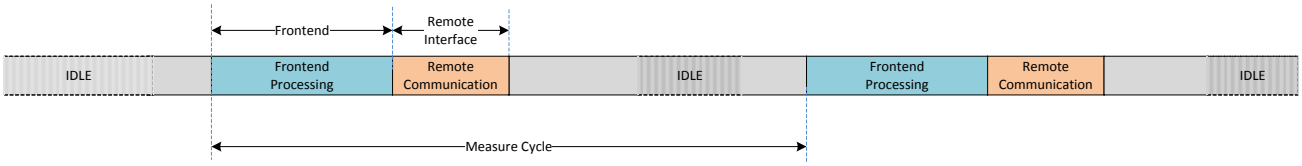
The following sections define the recommended sequences via remote interface for SPI.

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

4.2.1 Time Conversion Mode

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

Figure 18



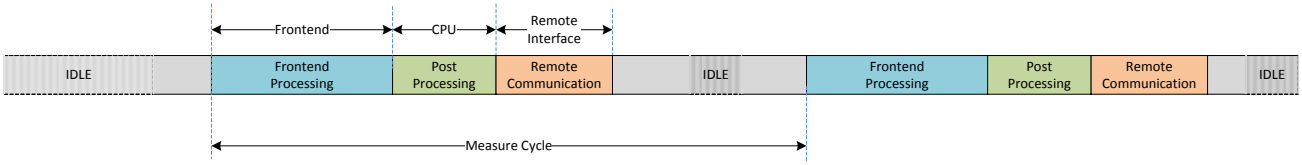
Step	Description	Opcodes & Data	Sequence
1	Check on interrupt **TSQ_FNS** (Task Sequencer finished) by reading **SRR_IRQ_FLAG**	0x7A 0xE0 *read data (Bit 0)*	Wait for interrupt
1a	Optionally read current task sequencer time from **SRR_TS_TIME** to check how much time is left for remote communication	0x7A 0xE9 *read data*	
2	Read **SRR_ERR_FLAG** to check if any error has been 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*	
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* *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)**

4.2.2 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 19



In the following sequence some register names in RAM area and their appropriate addresses are referenced to acam's firmware. For a more detailed description of acam's firmware and its flow metering routines, please refer to "TDC-GP30 Datasheet Vol. 2"

Step	Description	Opcodes & Data	Sequence
1	Check on interrupt **FW_IRQ_S** (FW Interrupt synchronized with task sequencer) by reading **SRR_IRQ_FLAG** if interrupt is controlled by firmware, typically in flow meter mode	0x7A 0xE0 *read data (Bit 4)*	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* *read data* …….	Get results
4	Clear interrupt flag register by sending **RC_IF_CLR**	0x8D	Completion
5	Clear FW error flags by writing 0x00000000 to **RAM_R_FW_ERR_FLAG**	0x5A 0x27 0x00000007	

***) 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 Miscellaneous Functions

5.1 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 special code should be written to register CR_WD_DIS.

Important register

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

5.2 Timestamp

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

Important registers

CR_CPM	0x0C5	**TSV_UPD_MODE**: Timestamp Update Mode
SHR_EXC	0x0DD	**TSV_UPD**: Updates Timestamp **TSC_CLR**: Clear Time Stamp Counter
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

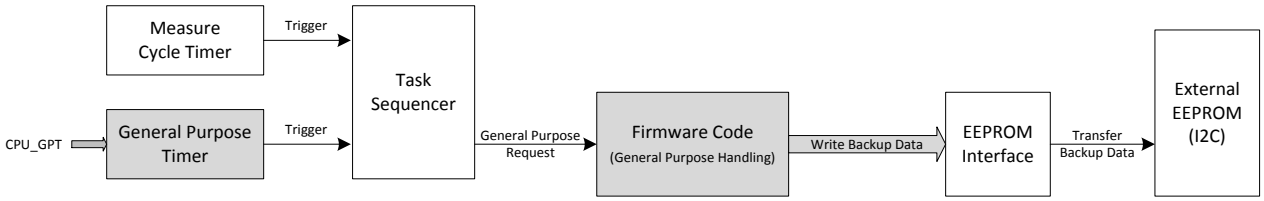
5.3 Backup Handling

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's 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.

Figure 20 EEPROM interface

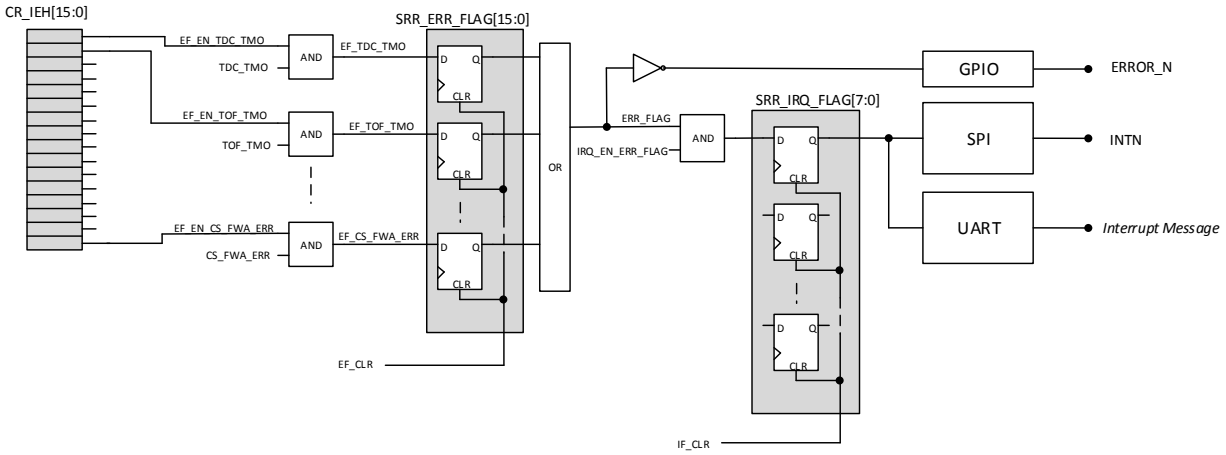


Important registers

CR_PI_E2P	0x0C1	**E2P_MODE**: Defines which GPIOs are used for I2C communication **E2P_ADR**: Address for external I2C EEPROM **E2P_PU_EN**: Enables internal pullups for I2C 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)

5.5 Error Handling

Figure 21



Important registers

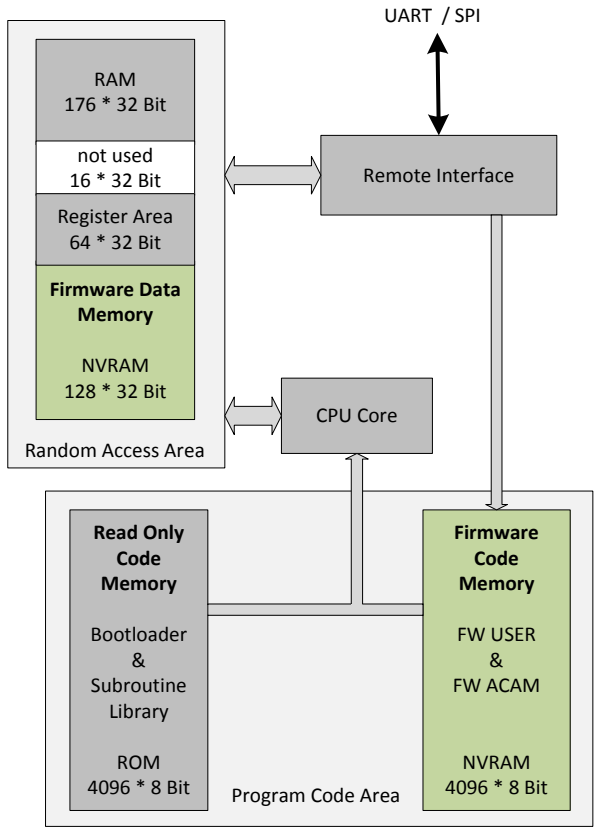
CR_IEH	0x0C4	**EF_EN_xx_xx_xx** (Bit 15..0): Enables Error Flags corresponding to register SRR_ERR_FLAG **IRQ_EN_ERR_FLAG**: Interrupt Request Enable for Error Flag
SHR_EXC	0x0DD	**IF_CLR**: Clears Interrupt Flag Register **EF_CLR**: Clears Error Flag Register
SRR_IRQ_FLAG	0x0E0	**ERR_FLAG**: Error Flag 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**

6 Handling Firmware with TDC-GP30

6.1 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 and one for the firmware data (e.g. parameters, configuration data).

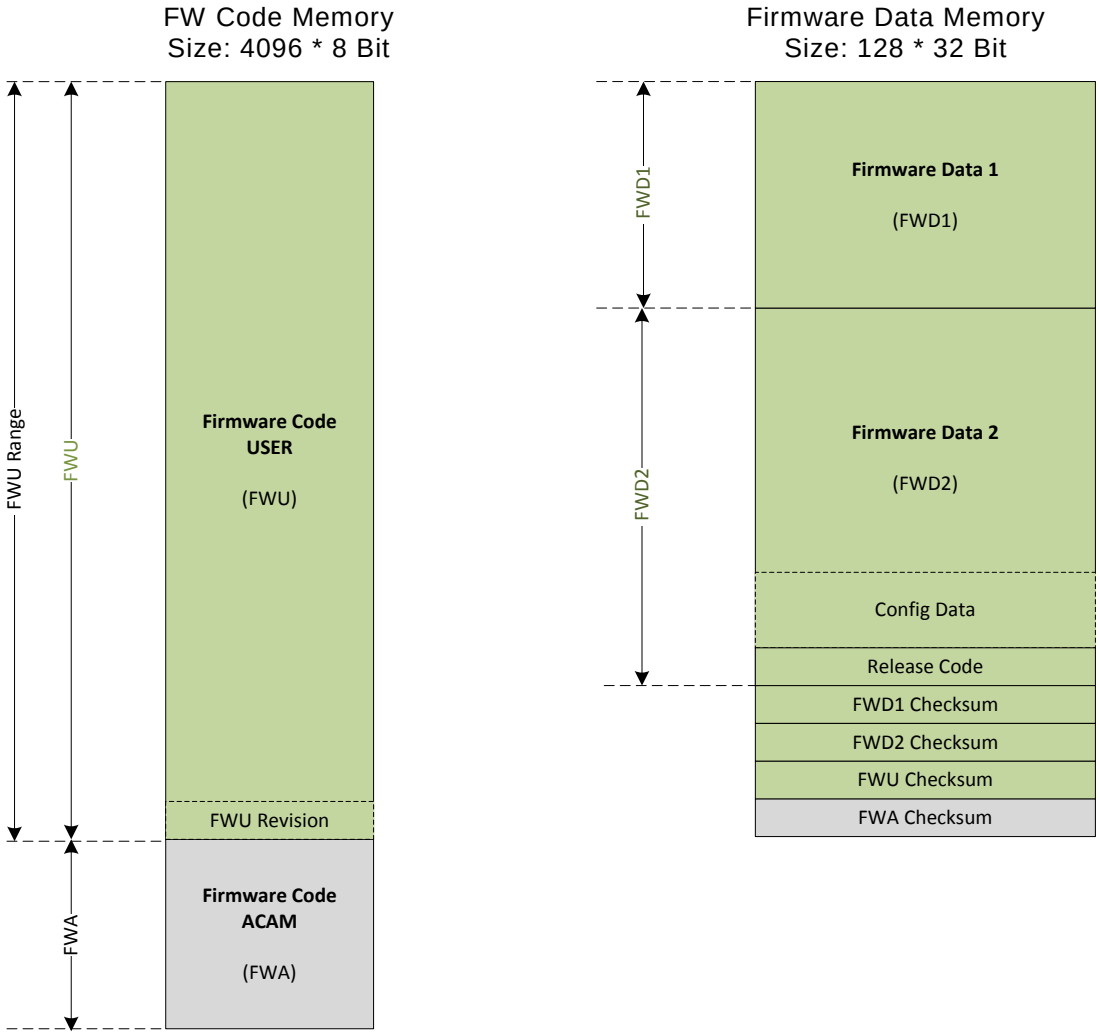
Figure 22



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

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

The FW program code includes a section in the upper-addressed area, which is pre-programmed by ACAM. This code contains trimming parameters for the NVRAMs and the analog part, as well some additional subroutine calls, supporting the access to subroutine library located in the **Read Only Code Memory**.



Green areas are intended for USER programming.

Important registers

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 ACAM can be read out from here

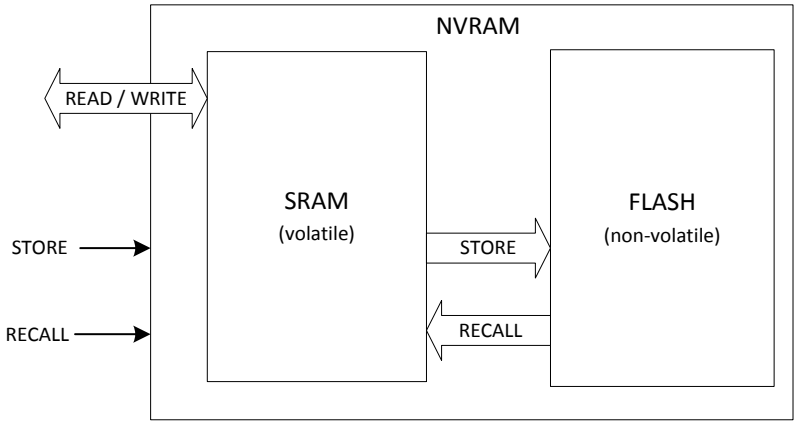
6.2 NVRAM Architecture

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.

Figure 23



Important registers

SHR_FW_TRANS_EN	0x0DF	Enables FW Transactions STORE or RECALL executed in **SHR_RC**
SHR_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**)

6.3 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.

Important FW Data

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 ACAM Code and its expected checksum is pre-programmed by ACAM 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 (Little Endian, block length byte, CRC, acknowledge).

6.3.1 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 0xXX 0xXX ….. 0x5C 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 *) 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	

*) **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)**

6.3.2 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 .. *) 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 0xXX 0xXX ….. 0x5C 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	

*) **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)**

6.3.3 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 *) 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 *) 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	

*) **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)**

6.4 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.

6.4.1 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 ACAM** (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:

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:

0x17C	**FWD1_CS_EXP**
0x17D	**FWD2_CS_EXP**
0x17E	**FWU_CS_EXP**

0x17F	**FWA_CS_EXP**

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:

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	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 ACAM 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	
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 ….. 0x5C 0x80 0x00 0x00 ….. …….	Clearing FW Code & FW Data

Step	Description	Opcodes & Data	Sequence
12	Write 0x00000000 to FWD addresses 0-127	0x5B 0x00 *) 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 *) *gen. checksum* 0x7B 0x7C *) *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* 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	

*) **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)**

6.4.2 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 acam for further support.

6.5 Firmware Lock & Erase

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

- A write protection for **Firmware Code**
- A read protection for F**irmware Data**
- A read protection for **configuration registers**

Please keep in mind, that F**irmware Code** is read protected all the time.

A firmware lock is performed by following steps:

- Make sure that desired F**irmware Code** & F**irmware 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)

Important registers

SHR_FW_ TRANS _EN	0x0DF	Enables FW Transactions STORE or RECALL executed in **SHR_RC**
SHR_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

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	
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
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	

*) **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 Appendix

7.1 Calculating Task Sequencer Cycle Time

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

MRCT	Task			Time [ms]
A	Refreshing Voltage Regulator		Always performed at start of cycle	1.7
	Frontend Processing	Max. time for ultrasonic measurement with a pause time of 20 ms	For a differentiated calculation please refer to corresponding chapter	21
	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			

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			

7.2 Glossary

CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
FEP	Frontend Processing
FWC	Firmware Code
FWD	Firmware Data
PP	Post Processing
RAA	Random Access Area
RAM	Random Access Memory
RI	Remote Interface
TOF	Time of Flight
UART	Universal Asynchronous Receiver & Transmitter
UFC	Ultrasonic Flow Converter

7.3 Document History

Revision	Remarks	Compiled	Released
0.1	Initial Release	HG, 2014-10-10	HG, 2014-10-10
0.2	Official release	HG,SK, 2015-07-01	NB 2015-07-01

member of the ams group

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