PGA900 Software

User's Guide

Literature Number: SLDU013A
May 2015–Revised January 2016

Contents

List of Figures

List of Tables

User's Guide
SLDU013A–May 2015–Revised January 2016

PGA900 Software User's Guide

1 Software Architecture

This section describes PGA900 software architecture. Figure 1 shows PGA900 software architecture.

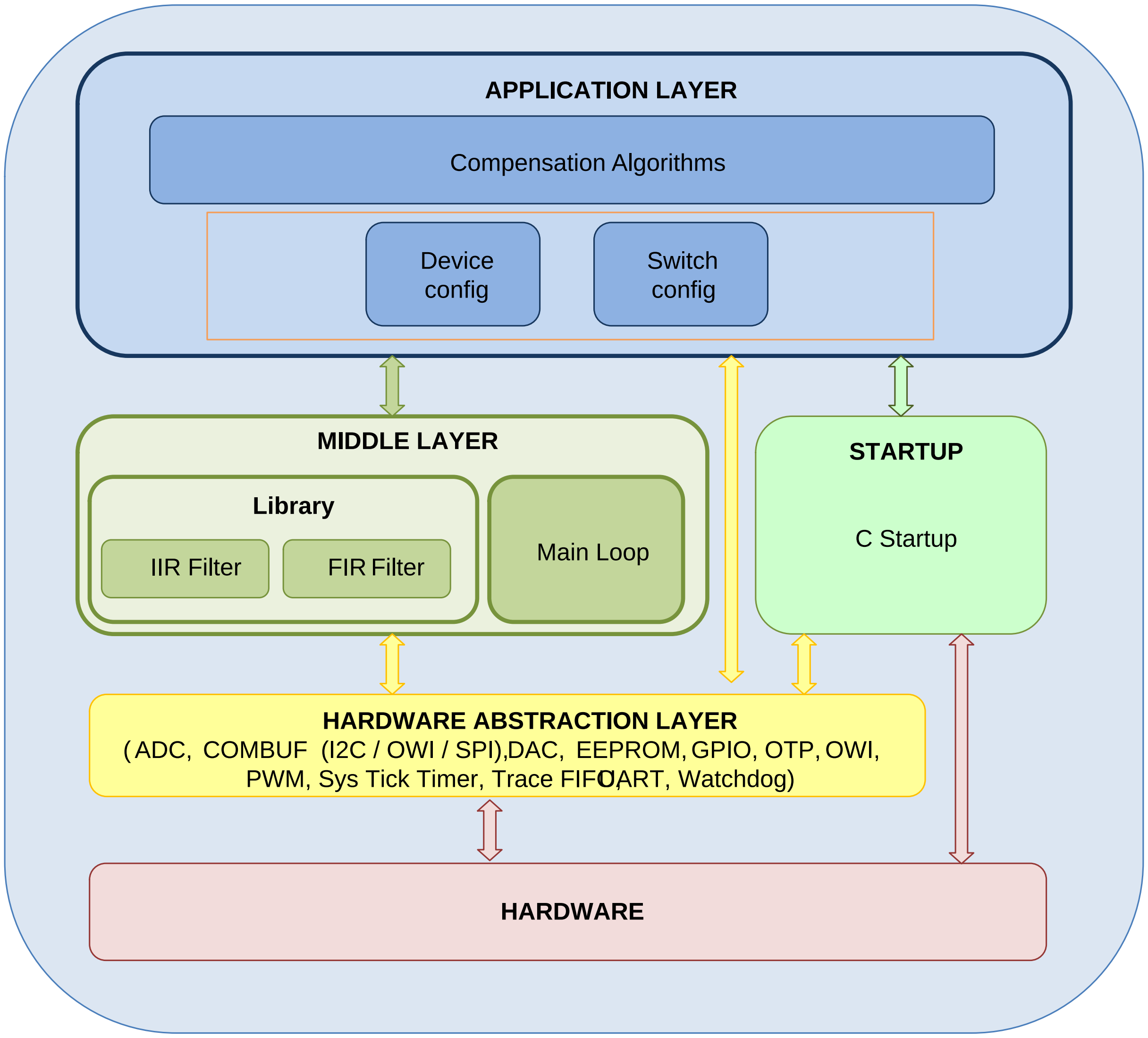


Figure 1. PGA900 Software Architecture

The PGA900 software consists of three layers, which are described as follows:

- Hardware abstraction layer (HAL): The HAL implements software that manipulates various PGA900 registers. The HAL register variables are in implemented in application program interfaces (APIs) that the application layer (APP) uses to access the PGA900 Control and Status registers.
- Middle layer (ML): The ML implements hardware-independent functionality including Filter routines and

C main function. The reference code includes example FIR filter routines.

- APP: The APP not only configures the PGA900 for specific user configuration, but also implements the compensation algorithms. The reference code includes one application, which demonstrates the use of peripheral drivers.

In addition, the PGA900 software also contains power-up routines that implement the following functions:

- Locate vector table
- Clear RAM

1.1 Source Code Directory Structure

This section describes the source code directory structure and source file details.

Figure 2 shows the PGA900 source code directory structure.

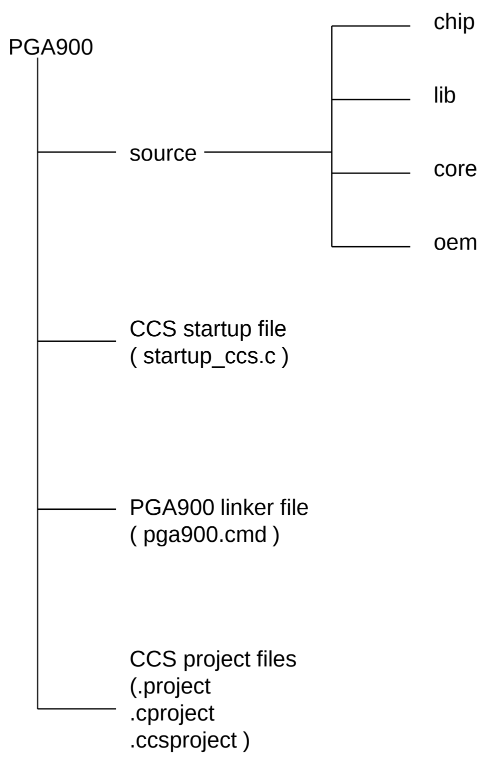


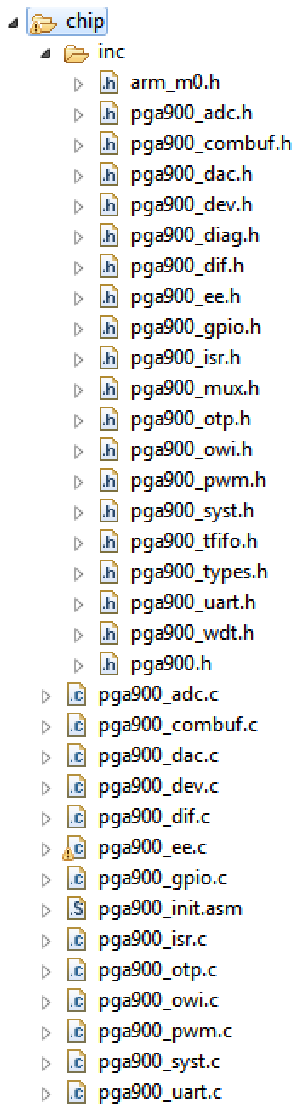
Figure 2. Source Code Directory Structure

- The source folder contains source files. Source files are arranged as per PGA900 software architecture.
  - The chip folder contains the definition of hardware registers and initialization and configuration of PGA900 hardware peripherals.
  - The lib folder contains IIR and FIR filter library functions.
  - The core folder contains main function.
  - The oem folder contains oem-defined/specific-application functions. Application functions use low level and ML functions.
- CCS startup file – Startup code for use with TI's Code Composer Studio.
- PGA900 linker file – pga900.cmd is a linker command file. The MEMORY directive defines the PGA900 memory configuration. The SECTIONS directive controls how sections are built and allocated.

1.1.1 List of Source Code Files

1.1.1.1 CHIP

The chip folder contains header files and source code files. Figure 3 is a screenshot of the chip folder.



```
chip
  inc
    arm_m0.h
    pga900_adc.h
    pga900_combuf.h
    pga900_dac.h
    pga900_dev.h
    pga900_diag.h
    pga900_dif.h
    pga900_ee.h
    pga900_gpio.h
    pga900_isr.h
    pga900_mux.h
    pga900_otp.h
    pga900_owi.h
    pga900_pwm.h
    pga900_syst.h
    pga900_tfifo.h
    pga900_types.h
    pga900_uart.h
    pga900_wdt.h
    pga900.h
  pga900_adc.c
  pga900_combuf.c
  pga900_dac.c
  pga900_dev.c
  pga900_dif.c
  pga900_ee.c
  pga900_gpio.c
  pga900_init.asm
  pga900_isr.c
  pga900_otp.c
  pga900_owi.c
  pga900_pwm.c
  pga900_syst.c
  pga900_uart.c
```

Figure 3. Source Code of CHIP Folder

Table 1. List of Source Files in CHIP Folder

Source Files	Description
pga900_adc.c and pga900_adc.h	Source file for analog-to-digital converter interface
pga900_combuf.c and combuf.h	Source file for COMBUF interface
pga900_dac.c and pga900_dac.h	Source file for digital-to-analog converter interface
pga900_dev.c and pga900_dev.h	Source file for PGA900 device
pga900_dif.c and pga900_dif.h	Source file for digital interface
pga900_ee.c and pga900_ee.h	Source file for EEPROM interface
pga900_gpio.c and pga900_gpio.h	Source file for GPIO interface
pga900_isr.c and pga900_isr.h	Source file for interrupt subroutine
pga900_otp.c and pga900_otp.h	Source file for OTP interface
pga900_owi.c and pga900_owi.h	Source file for OWI interface
pga900_pwm.c and pga900_pwm.h	Source file for PWM interface
pga900_syst.c and pga900_syst.h	Source file for sys tick timer interface
pga900_uart.c and pga900_uart.h	Source file for UART interface
pga900.h	Definition of PGA900 Control and Status register
pga900_mux.h	Source file for analog and digital multiplexer selection
pga900_tfifo.h	Source file for Trace FIFO interface
pga900_wdt.h	Source file for Watchdog interface
pga900_types.h	Data types are type defined
arm_m0.h	ARM Cortex M0 register definitions
pga900_diag.h	Diagnostic bits are defined. The three diagnostic registers are PSMON1, PSMON2, and AFEDIAG.
pga900_init.asm	System initialization file The _c_int00 assembly function performs: • Switch to user mode • Sets up the initial value of the stack pointer (SP) • Calls the function _ _TI_auto_init to perform the C auto initialization • Calls the function main to run the C program • Two compile time switches are available SEC_LOCK_TESTING – .set 0 /* Enables/disable security lock at power on */ DATA_RAM_MBIST_TESTING – .set 0 /* Enable/disable Data RAM memory built in self test */ By default, these compile time switches are off.

1.1.1.2 Lib

The lib folder contains library header and source files. Figure 4 is a screenshot of the lib folder.

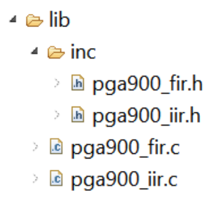


Figure 4. Source Code of Lib Folder

Table 2. List of Source Files in Lib Folder

Source Files	Description
pga900_fir.c and pga900_fir.h	Source files for FIR filter
pga900_iir.c and pga900_iir.c	Source files for IIR filter

1.1.1.3 Core

The core folder contains main header file and source file. Figure 5 is a screenshot of the core folder.

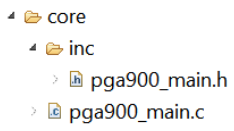

core
inc
pga900_main.h
pga900_main.c

Figure 5. Source Code of Core Folder

Table 3. List of Source Files of Core folder

Source Files	Description
pga900_main.c and pga900_main.h	Source files for main function

1.1.1.4 Oem

The oem folder contains application-specific header and source files. Figure 6 is a screenshot of the oem folder.

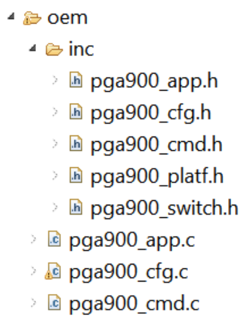

oem
inc
pga900_app.h
pga900_cfg.h
pga900_cmd.h
pga900_platf.h
pga900_switch.h
pga900_app.c
pga900_cfg.c
pga900_cmd.c

Figure 6. Source Code of Oem Folder

Table 4. List of Source Files of Oem Folder

Source Files	Description
pga900_app.c and pga900_app.h	Source files for application – Compensation algorithms are implemented
pga900_cmd.c and pga900_cmd.h	Source files to process commands
pga900_cfg.c and pga900_cfg.h	Source files to configure PGA900 peripherals
pga900_platf.h	All header files are included
pga900_switch.h	Definition of compile time switches

This section describes the configuration of the PGA900 software.

1.2 Software Configurations

1.2.1 Device Configuration

PGA900 peripherals can be configured using pga900_cfg.c source file. The configuration involves writing specific values to registers.

The device configuration file, pga900_cfg.c, contains the CFG_Peripheral_Config() function. This function can be modified to configure PGA900 peripherals.

Table 5 shows the list of PGA900 peripheral configuration functions.

Table 5. List of PGA900 Peripheral Configuration Functions

Device Module	Configuration Function Name	Description

ADC	ADC_Pchannel_Config()
	ADC_Tchannel_Config()
	ADC_Config()
		Configuration of pressure and temperature ADC

COMBUF	COMBUF_RX_INT_ENABLE()	Enable COMBUF receive interrupt
DAC	DAC_Config()	Configure DAC device module
PGA900 device clock	M0_ConfigClock()	Configure PGA900 clock
Digital interface	DIGITAL_Interface_Config()	Configure digital interfaces (SPI, I^{2}C, and OWI)
EEPROM	N/A	This module does not require the configuration function.
GPIO	GPIO_Config()	Configure GPIO direction
ISR	N/A	This module does not require the configuration function.

Digital multiplexer	TOPDIG_MUX_CONFIG()
	TONDIG_MUX_CONFIG()
		Configure digital multiplexer

Analog multiplexer	AMUX_TIN_MUX_CONFIG()
	AMUX_TOUT_MUX_CONFIG()
		Configure analog multiplexer

OTP	N/A	This module does not require the configuration function.
OWI	OWI_Config()	Configure one-wire interface
PWM	N/A	This module does not require the configuration function.
System tick timer	SYST_Config()	Configure system tick timer
Trace FIFO	TRACE_FIFO_CONFIG()	Configure trace FIFO Control and Status register
UART	UART_Config()	Configure UART's baud rate, data bits, parity, and stop bits
Watchdog timer	N/A	This module does not require the configuration function.

Refer to *Section 6* for more information. Refer to the *Functional Block Diagram* section of the PGA900 data sheet for more information about PGA900 peripherals, SLDS209.

1.2.2 Conditional Compilation Switches

The user can compile in or compile out PGA900 peripheral functionality by configuring various compile time switches available in the pga900_switch.h file. Compile only the necessary functions to optimize code size.

Table 6 shows list of conditional compiler switches.

Table 6. List of Conditional Compiler Switches

Compile Switch	Description
SYST_TESTING	When SYST_TESTING compile switch is 1, then the source code of system timer SysTick module available into pga900_syst.c and pga900_syst.h gets compiled. When SYST_TESTING compile switch is 0, then the source code of system timer SysTick module available into pga900_syst.c and pga900_syst.h gets suppressed.
DIAGNOSTIC_TESTING	When DIAGNOSTIC _TESTING compile switch is 1, then the source code of diagnostic module available into pga900_diag.h gets compiled. When DIAGNOSTIC _TESTING compile switch is 0, then the source code of diagnostic module available into pga900_diag.h gets suppressed.
EEPROM_TESTING	When EEPROM_TESTING compile switch is 1, then the source code of EEPROM module available into pga900_ee.c and pga900_ee.h gets compiled. When EEPROM_TESTING switch is 0, then the source code of EEPROM module available into pga900_ee.c and pga900_ee.h gets suppressed.
GPIO_TESTING	When GPIO_TESTING switch is 1, then the source code of GPIO module available into pga900_gpio.c and pga900_gpio.h gets compiled. When GPIO_TESTING switch is 0, then the source code of GPIO module available into pga900_gpio.c and pga900_gpio.h gets suppressed.
TEMP_MUX_CTRL_TESTING	When TEMP_MUX_CTRL_TESTING switch is 1, then the source code of temperature multiplexer available into pga900_mux.h gets compiled. When TEMP_MUX_CTRL_TESTING switch is 0, then the source code of temperature multiplexer available into pga900_mux.h gets suppressed.
ANALOG_TEST_OUT_TESTING	When ANALOG_TEST_OUT_TESTING switch is 1, then the source code of analog test out available into pga900_mux.h gets compiled. When ANALOG_TEST_OUT_TESTING switch is 0, then the source code of analog test out available into pga900_mux.h gets suppressed.
OTP_TESTING	When OTP_TESTING switch is 1, then the source code of OTP module available into pga900_otp.c and pga900_otp.h gets compiled. When OTP_TESTING switch is 0, then the source code of OTP module available into pga900_otp.c and pga900_otp.h gets suppressed.
OWI_TESTING	When OWI_TESTING switch is 1, then the source code of OWI module available into pga900_owi.c and pga900_owi.h gets compiled. When OWI_TESTING switch is 0, then the source code of OWI module available into pga900_owi.c and pga900_owi.h gets suppressed.
PWM_TESTING	When PWM_TESTING switch is 1, then the source code of PWM module available into pga900_pwm.c and pga900_pwm.h gets compiled. When PWM_TESTING switch is 0, then the source code of PWM module available into pga900_pwm.c and pga900_pwm.h gets suppressed.
TRACE_FIFO_TESTING	When TRACE_FIFO_TESTING switch is 1, then Trace FIFO related functions get compiled. When TRACE_FIFO_TESTING switch is 0, then Trace FIFO related functions get suppressed.
UART_TESTING	When UART_TESTING switch is 1, then the source code of UART module available into pga900_uart.c and pga900_uart.h gets compiled. When UART_TESTING switch is 0, then the source code of UART module available into pga900_uart.c and pga900_uart.h gets suppressed.
WATCH_DOG_TIMER_TESTING	When WATCH_DOG_TIMER_TESTING switch is 1, then the source code of watch dog module available into pga900_wdt.h gets compiled. When WATCH_DOG_TIMER_TESTING switch is 0, then the source code of watch dog module available into pga900_wdt.h gets suppressed.
COMBUF_CMD_TESTING	When COMBUF_CMD_TESTING switch is 1, then the source code available into pga900_cmd.c and pga900_cmd.h gets compiled. When COMBUF_CMD_TESTING switch is 0, then the source code available into pga900_cmd.c and pga900_cmd.h gets suppressed.
ADC_24_BIT_TESTING	This compile time switch should be OFF. 24 bit ADC read logic is implemented under this flag into pga900_adc.c source file. 24 bit ADC compensation algorithm is not supported.

Table 6. List of Conditional Compiler Switches (continued)

Compile Switch	Description
GATE_DRIVE_TESTING	When GATE_DRIVE_TESTING switch is 1, then gate drive is enabled. When GATE_DRIVE_TESTING switch is 0, then gate drive is disabled.

The user needs to modify these switches based on needed functionality and compile the code.

Usage example:

GPIO_TESTING compile time switch is set to 1 using the following syntax:

#define GPIO_TESTING 1

GPIO_TESTING compile time switch is set to 0 using the following syntax:

#define GPIO_TESTING 0

By default, the following compile time switches are enabled and others are disabled.

#define DIAGNOSTIC_TESTING ON

#define ANALOG_TEST_OUT_TESTING ON

1.2.3 Linker Configuration

The linker file defines various memory sections and places code and data into these sections in target memory.

The pga900.cmd is a CCS linker configuration file for PGA900. The pga900.cmd uses two directives: MEMORY and SECTIONS.

1.2.3.1 MEMORY

The MEMORY directive allows the user to specify a model of target memory. PGA900 contains 8KB of OTP memory, 1KB of data SRAM and 8KB of development RAM.

Figure 7 shows memory map of PGA900.

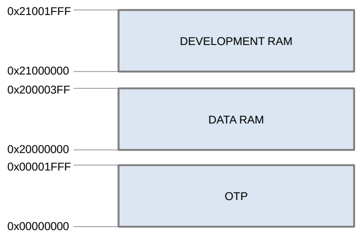


Figure 7. Memory Map of PGA900

1.2.3.2 SECTIONS

The SECTIONS directive controls how sections are built and allocated.

The ARM C/C++ compiler automatically generates the following sections:

- Initialized sections: .text, .const, .cinit, and so forth
- Uninitialized sections: .bss, .stack, .sysmem, and so forth

Using the SECTIONS directive, place these generated sections into memory.

See the *ARM Assembly Language Tools v4.6 User's Guide*, SPNU118, for more information about the MEMORY and SECTIONS directive.

Code from the pga900.cmd file follows:

```
#define APP_BASE 0x00000000
#define RAM_BASE 0x20000000

/* System memory map */
MEMORY
{
    /* Application stored in and executes from internal flash */
OTP (RX) : origin = APP_BASE, length = 0x00002000
    /* Application uses internal RAM for data */
SRAM (RWX) : origin = 0x20000000, length = 0x00000400
    /* Application can be stored in DRAM and executes from DRAM */
DRAM (RX) : origin = 0x21000000, length = 0x00002000
}
/* Section allocation in memory */
SECTIONS
{
    .intvecs: > APP_BASE
    .text : > OTP
    .const : > OTP
    .cinit : > OTP
    .pinit : > OTP
    .init_array : > OTP
    .vtable : > RAM_BASE
    .data : > SRAM
    .bss : > SRAM
    .sysmem : > SRAM
    .stack : > 0x20000000
}
```

The PGA900 device contains 8KB of development RAM. Certain sections can be configured to run from development RAM.

The following example shows how to run a certain section from development RAM:

- CODE_SECTION pragma allocates space for the symbol in C. Example:

```
#pragma CODE_SECTION (PADC_Handler, ".ram_exec_sect")
interrupt void PADC_Handler(void)
{
….
….
}
```

  In the previous example, CODE_SECTION pragma allocates space for the symbol PADC_Handler.

- Modify SECTIONS part of the pga900.cmd file as follows:

```
/* Section allocation in memory */
SECTIONS
{
.intvecs: > APP_BASE
.text : > OTP
.const : > OTP
.cinit : > OTP
.pinit : > OTP
.binit : > OTP
.init_array : > OTP
.vtable : > RAM_BASE
.data : > SRAM
.bss : > SRAM
.sysmem : > SRAM
.stack : > 0x20000000
.ram_exec_sect : load = OTP, run = DRAM, table(BINIT)
}
```

  The last statement in the SECTIONS part indicates that ram_exec_sect section is loaded into OTP and runs from DRAM. That is, PADC_Handler function is loaded into OTP and runs from development RAM.

2 Software Development Tools and Build Environment

This section describes the software development tools version and compiler option used for PGA900 code development.

- For PGA900 software development, Code Composer Studio Version: 6.1.1.00022 is used.
- The current version of the PGA900 code is built with ARM TI v5.2.6 compiler using EABI object file.
- Compiler options control the operation of the compiler such as selecting the ARM processor version, code optimization, and debugging.

  The following compiler options are selected for PGA900 code development:

```
-mv6M0
--code_state=16
--abi=eabi
-me
-O4
--opt_for_speed=5
--fp_mode=relaxed
-g
--optimize_with_debug=on
--diag_warning=225
--display_error_number
--optimizer_interlist
--call_assumptions=0
--single_inline
--gen_opt_info=2
-k
--asm_listing
```

See *ARM Optimizing C/C++ Compiler v4.6 User's Guide*, SPNU151, for more information about compile switches.

Code Composer Studio Version: 6.1.1.00022 can be downloaded from: http://processors.wiki.ti.com/index.php/Download_CCS

3 Application Software

This section provides a detailed description of an existing reference sample application and how the user can build their application using the existing reference software.

3.1 Sample Application Software

The reference code includes a sample application, which demonstrates the compensation algorithm. This section describes the sample compensation algorithm.

3.1.1 Compensation Equation

The second-order TC compensation equation is as follows:

$$\begin{aligned} \text{DAC} = &\left(h_0 + h_1\text{TADC} + h_2\text{TADC}^2\right) \\ &+ \left(g_0 + g_1\text{TADC} + g_2\text{TADC}^2\right)\text{PADC} \\ &+ \left(n_0 + n_1\text{TADC} + n_2\text{TADC}^2\right)\text{PADC}^2 \end{aligned}$$

where

- P is the PADC value.
- T is the TADC value. (1)

3.1.2 Normalization of ADC Values

Normalize as follows:

Pn = PADC / 2^{14} (2)

Tn = TADC / 2^{14} (3)

Dn = DAC / 2^{14} (4)

The compensation equation can be written as:

$$\begin{aligned} D_n = &\left(h_{0n} + h_{1n}T_n + h_{2n}T_n{}^2\right) \\ &+ \left(g_{0n} + g_{1n}T_n + g_{2n}T_n{}^2\right)P_n \\ &+ \left(n_{0n} + n_{1n}T_n + n_{2n}T_n{}^2\right)P_n{}^2 \end{aligned} \quad (5)$$

3.1.3 Scaling of Coefficient

Store coefficients in EEPROM by multiplying the floating point version by 2^{14}.

h_{0EE} = round(h0n × 2^{14})
h_{1EE} = round(h1n × 2^{14})
h_{2EE} = round h2n × 2^{14})
g_{0EE} = round(g0n × 2^{14})
g_{1EE} = round(g1n × 2^{14})
g_{2EE} = round(g2n × 2^{14})
n_{0EE} = round(n0n × 2^{14})
n_{1EE} = round(n1n × 2^{14})
n_{2EE} = round(n2n × 2^{14})

NOTE: Choose the normalization such that each coefficient is 2 bytes in width in order to fit each coefficient in 2 bytes in the EEPROM.

3.1.4 Algorithm and Implementation

3.1.4.1 Algorithm

Based on normalized ADC values and scaled coefficient values, the algorithm is as follows:

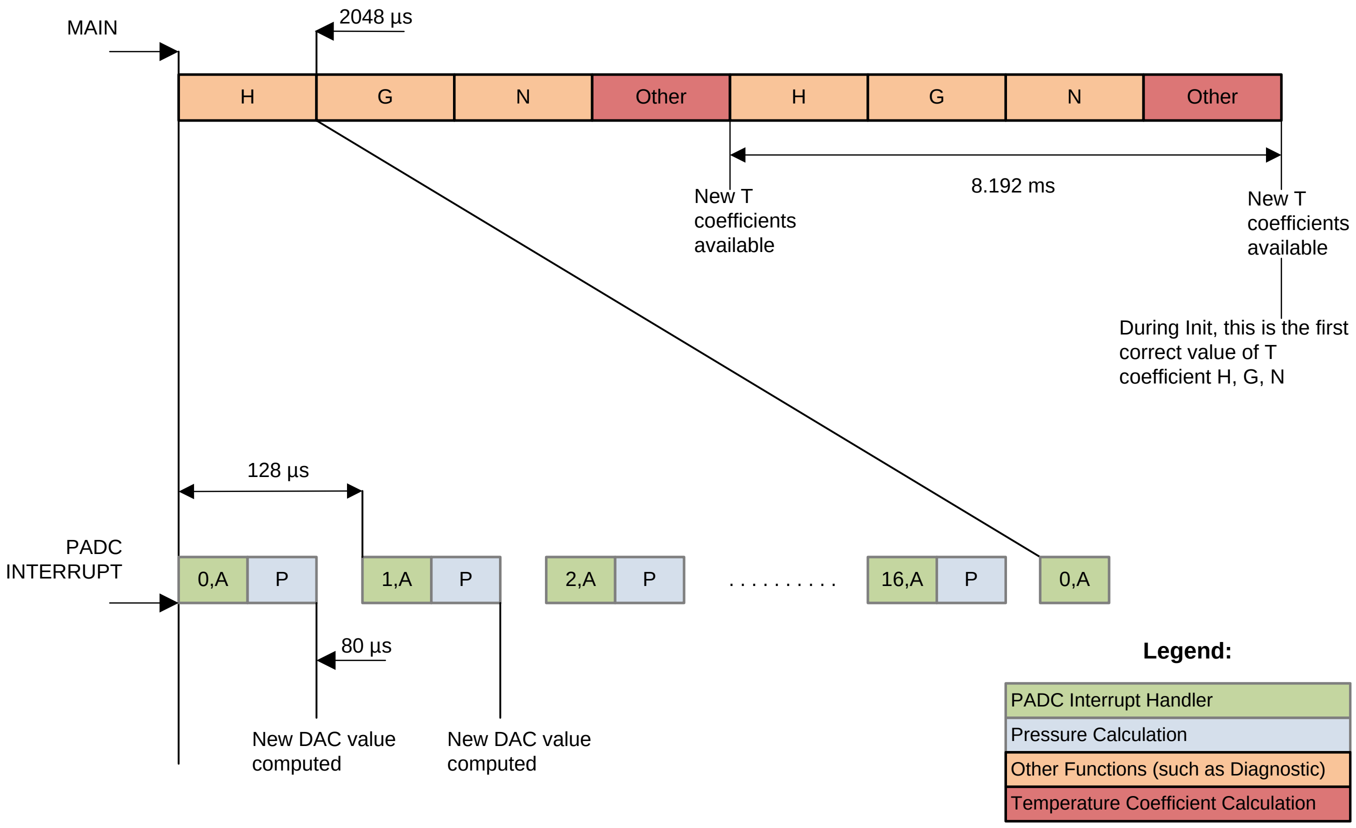
$$D_n = \left(\frac{h_{0EE}}{2^{14}} + \frac{h_{1EE}}{2^{14}}\frac{TADC}{2^{14}} + \frac{h_{2EE}}{2^{14}}\left(\frac{TADC}{2^{14}}\right)^2\right)$$
$$+ \left(\frac{g_{0EE}}{2^{14}} + \frac{g_{1EE}}{2^{14}}\frac{TADC}{2^{14}} + \frac{g_{2EE}}{2^{14}}\left(\frac{TADC}{2^{14}}\right)^2\right)\left(\frac{PADC}{2^{14}}\right)$$
$$+ \left(\frac{n_{0EE}}{2^{14}} + \frac{n_{1EE}}{2^{14}}\frac{TADC}{2^{14}} + \frac{n_{2EE}}{2^{14}}\left(\frac{TADC}{2^{14}}\right)^2\right)\left(\frac{PADC}{2^{14}}\right)^2$$
$$DAC = D_n 2^{14} \tag{6}$$

3.1.4.2 Implementation

The pseudocode for implementing the previous algorithm is:

```
1. $h = h_{2EE} \times TADC$
2. $h = h>>14$
3. $h = h + h_{1EE}$
4. $h = h \times TADC$
5. $h = h>>14$
6. $h = h + h_{0EE}$
7. $g = g_{2EE} \times TADC$
8. $g = g>>14$
9. $g = g + g_{1EE}$
10. $g = g \times TADC$
11. $g = g>>14$
12. $g = g + g_{0EE}$
13. $n = n_{2EE} \times TADC$
14. $n = n>>14$
15. $n = n + n_{1EE}$
16. $n = n \times TADC$
17. $n = n>>14$
18. $d = n \times PADC$
19. $d = d>>14$
20. $d = d + g$
21. $d = d \times PADC$
22. $d = d>>14$
23. $d = d + h$
24. $d = d$
```

3.1.4.3 Calculation Timing


MAIN
2048 µs
H
G
N
Other
H
G
N
Other
8.192 ms
New T coefficients available
New T coefficients available
During Init, this is the first correct value of T coefficient H, G, N
128 µs
PADC INTERRUPT
0,A
P
1,A
P
2,A
P
. . . .
16,A
P
0,A
80 µs
New DAC value computed
New DAC value computed
Legend:
PADC Interrupt Handler
Pressure Calculation
Other Functions (such as Diagnostic)
Temperature Coefficient Calculation

A All calculations are integer math.

B PADC handler runs every 128 µs.

C DAC register is written in PADC handler.

D A new DAC value is available every 128 µs.

E FIR filter for T.

F Second order TC compensation for offset, gain, and non-linearity.

G Diagnostic checks run every 8.192 ms.

Figure 8. Calculation Timing Diagram

The P ADC is set up for a 128-µs interrupt. The DAC calculations are completed in 1-ADC interrupts. This gives a DAC update rate of 128 µs. Note that the TADC values are read every 128 µs resulting in 64 samples accumulated using FIR filter. The free slots can be used to implement filters and diagnostics.

3.1.5 EEPROM

To store coefficients into EEPROM, EEPROM_Default_Tcoeff[] buffer is present in pga900_ee.c. The calculated coefficients are stored in the EEPROM_Default_Tcoeff[] buffer.

User can modify these values based on their algorithm. User has to write correct CRC into the last location of the EEPROM as well as update following macro with correct CRC. #define EEPROM_USER_CRC 0x78 present into pga900_cfg.c.

The following flow chart in Figure 9 indicates logic developed for writing default values. Refer to pga900_cfg.c.

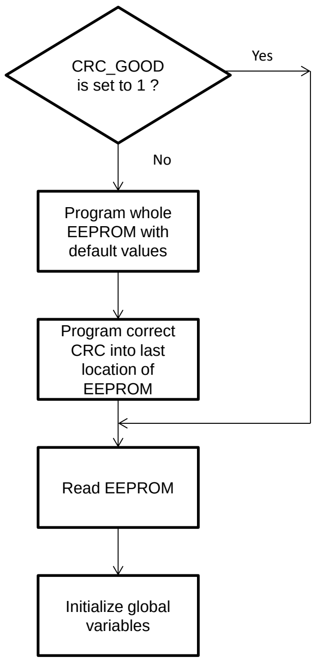

CRC_GOOD is set to 1 ?
Yes
No
Program whole EEPROM with default values
Program correct CRC into last location of EEPROM
Read EEPROM
Initialize global variables

Figure 9. Logic for Writing Default Values

3.2 Building Application Software

The user can build their application software using two approaches.

3.2.1 Addition of Application Functions

The user can develop their application software by developing new functions or modifying the functions that are available in the reference sample application. The reference sample application is available in pga900_app.c and pga900_app.h. New functions can be added into the existing sample application source files, pga900_app.c and pga900_app.h.

Call new developed functions from the APP_Calculate_Coeff() function available into pga900_app.c.

NOTE: APP_Calculate_Coeff() contains the state machine. In the state machine, function pointers are used.

User can configure the desired application-specific peripheral of PGA900 by modifying the CFG_Peripheral_Config() function available into the source file pga900_cfg.c.

3.2.2 Addition of Application Source Files

The user can add application-specific new source files into the oem folder. The user needs to create source file (.c file) and add into the oem folder. Similarly, the header file must be created (if required) and added into oem\inc folder. Include the new header file into the pga900_platf.h.

Call the new developed functions from APP_Calculate_Coeff() function available into pga900_app.c.

NOTE: APP_Calculate_Coeff() contains the state machine. In the state machine, function pointers are used.

4 Coding Standards

For PGA900 code development, use standardize common code development practices. A summary of coding standards follow:

- A line of code shall not exceed 80 characters in length.
- One level of indentation of code shall equal four character spaces.
- The tab character shall not appear in source code.
- Braces follow a loop or a conditional construct even if there is only one statement.
- Each source file and header file includes a file header.
- Each function includes a function header that states the purpose of the function and provides documentation for the function. This documentation includes: a full description of each parameter, the return type, all possible return values, required preconditions and/or defined post conditions, and a reference to related functions.

Example:

```
/*================================================================= */
/**
* ADC_Pchannel_Config() Configure P channel ADC.
*
* @param pconfigValue Data to be written to PADC config register.
*
* @param pgainValue Data to be written to P channel gain select register.
*
* @return none
*
* @post none
*
* @see ADC_Tchannel_Config
*/
/* ================================================================= */
```

- The file name of a source file indicates the function of the file (for example, pga900_adc.c).
- All external API functions are prefixed with the module name followed by an underscore (for example, ADC_Pchannel_Config()).
- All local functions are named consistently. The local functions naming convention should be different from the convention of global functions/external APIs. Example:
  - ADC_Pchannel_Config() – Global function
  - processCmd1() – Local function
- Variable names are consistent. Example:
  - ADC_PchannelValue – Global variable
  - cmdNum – Local variable
- Include three blank lines between the closing brace of one function and the start of the next function or the function comment block of the next function.
- If the function is properly documented in a header file, that description may be called by placing a documentation comment opening, a space, "@fn", a space, and the function name followed by parentheses. In this case, a short description is required in the source file function header, but the source file may omit the remainder of the parameters except "@see", which names the header file holding the function declaration. (This is to detect and correct errors in documentation linkage.)

Example: The ADC_Pchannel_Config() function is properly documented in the header file.

```
/* ========================================================================== */
/**
* ADC_Pchannel_Config() Configure P channel ADC.
*
* @param pconfigValue Data to be written to PADC config register.
*
* @param pgainValue Data to be written to P channel gain select register.
*
* @return none
*
* @post none
*
* @see ADC_Tchannel_Config
*/
/* ========================================================================== */
```

Source file omits the remaining parameters.

```
/* ========================================================================== */
/**
* @fn ADC_Pchannel_Config() Configure P channel ADC.
*
* @see pga900_adc.h
*/
/* ========================================================================== */
void ADC_Pchannel_Config(UC pconfigValue, UC pgainValue)
{
P_GAIN_SELECT = pgainValue;
PADC_CONFIG = pconfigValue;
}
```

5 Types

The following types shown in Table 7 are defined in the pga900_types.h.

Table 7. Type Definitions

Type	Type Definition	Purpose
SC	typedef signed char	Signed 8-bit integer
UC	typedef unsigned char	Unsigned 8-bit integer
S2	typedef signed short	Signed 16-bit integer
US	typedef unsigned short	Unsigned 16-bit integer
SL	typedef signed long	Signed 32-bit integer
UL	typedef unsigned long	Unsigned 32-bit integer
SI	typedef signed int	Signed 32-bit integer
UI	typedef unsigned int	Unsigned 32-bit integer
FLOAT	typedef float	32-bit integer
VSC	typedef volatile signed char	Volatile signed 8-bit integer
VUC	typedef volatile unsigned char	Volatile unsigned 8-bit integer
VS2	typedef volatile signed short	Volatile signed 16-bit integer
VUS	typedef volatile unsigned short	Volatile unsigned 16-bit integer
VSL	typedef volatile signed long	Volatile signed 32-bit integer
VUL	typedef volatile unsigned long	Volatile unsigned 32-bit integer
VSI	typedef volatile signed int	Volatile signed 32-bit integer
VUI	typedef volatile unsigned int	Volatile unsigned 32-bit integer
VFLOAT	typedef volatile float	Volatile 32-bit integer

6 Functions

6.1 API Functions

This section describes the API functions of all hardware interfaces, the library, main loop, and application.

NOTE:

- API names that appear in capital letters are macros.
- The device modules that have global variables associated with them, contain a reset initialization function. Some of the device modules do not have global variables; such device modules reset initialization functions are not defined.

6.2 ADC Functions

This section describes API functions of the ADC interface.

Table 8. ADC Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
void ADC_Config (UC configValue1)	Configure ADC channels	configValue1 Data to be written to ADC_CFG_1 register.	None	None
void ADC_Pchannel_Config (UC pconfigValue, UC pgainValue)	Configure P channel ADC	pconfigValue Data to be written to PADC config register. pgainValue Data to be written to P channel gain select register.	None	None
void ADC_Tchannel_Config (UC tconfigValue, UC tgainValue)	Configure T channel ADC See also ADC_Pchannel_Config	tconfigValue Data to be written to TADC config register. 3tgainValue Data to be written to T channel gain select register.	None	None
interrupt void PADC_Handler (void)	Interrupt subroutine for P channel ADC. Read P channel data is stored into variable ADC_PchannelValue and T channel data into variable ADC_TchannelValue.	None	None	None
interrupt void TADC_Handler (void)	Interrupt subroutine for T channel ADC. Read T channel data is stored into variable ADC_TchannelValue.	None	None	None
void ADC_Reset_Init (void)	Initializes all ADC related global variables	None	None	None
ADC_PCHANNEL_DISABLE()	Disables PADC decimator	None	None	None
ADC_PCHANNEL_ENABLE ()	Enables PADC decimator	None	None	None
ADC_TCHANNEL_DISABLE ()	Disables TADC decimator	None	None	None
ADC_TCHANNEL_ENABLE()	Enables TADC decimator	None	None	None
CONFIG_ALPWR(x)	Configure ALPWR register	x - Value to be written to ALPWR register.	None	None

6.3 COMBUF Functions

This section describes API functions of COMBUF interface.

Table 9. COMBUF Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
interrupt void COMBUF_Handler (void)	Interrupt subroutine for COMBUF	None	None	None
void COMBUF_Reset_Init (void)	Initializes all COMBUF related global variables	None	None	None
COMBUF_RX_INT_DISABLE ( )	Disable COMBUF receive interrupt	None	None	None
COMBUF_RX_INT_ENABLE ( )	Enable COMBUF receive interrupt	None	None	None
COMBUF_RX_STATUS_CLEAR ( )	Clear COMBUF receive status bit by writing '1'	None	None	None

6.4 DAC Functions

This section describes API functions of DAC interface.

Table 10. DAC Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
void DAC_Config (UC dacconfigValue, UC dacmodValue, UC dacgainValue)	Configure DAC	dacconfigValue Data to be written to DAC control register. dacmodValue Data to be written to DAC configuration register. dacgainValue Data to be written to OP_STAGE_CTRL register.	None	None
void DAC_Loopback_Config (UC lpconfigValue)	Configure DAC loop-back	lpconfigValue Data to be written to loop-back control register.	None	None
void DAC_Reset_Init (void)	Initializes all DAC related global variables	None	None	None
DAC_REG0_CONFIG(x)	Configure DAC_REG0	x 14-bit value	None	None
DAC_REG1_CONFIG(x)	Configure DAC_REG1	x 14-bit value	None	None

6.5 Microcontroller Functions

This section describes API functions of PGA900 device specific functions.

Table 11. Microcontroller Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
void M0_ConfigClock (UC value)	Configure Cortex M0 clock	value Data to be written to clock control register	None	None
MICRO_INTERFACE_CONFIG(x)	Configure Micro interface control register	x - value 0x00 to 0x07	None	None
BRDG_CTRL_CONFIG(x)	Configure BRDG_CTRL register	x - Value to be written to BRDG_CTRL register.	None	None

6.6 Code Security Functions

Table 12. Code Security Functions

Function Name	Description	Parameters	Returns	Postcondition
SEC_LOCK_ENABLE ( )	Enable security lock	None	None	None
SEC_LOCK_DISABLE ( )	Disable security lock	None	None	None

6.7 Memory Built-In Self-Test Functions

Table 13. Memory Built-In Self-Test Functions

Function Name	Description	Parameters	Returns	Postcondition
MBIST_CONFIG(x)	Configure RAM built-in memory test	x - Value to be written to RAMBIST_CONTROL register	None	None
MBIST_STATUS( )	Get the MBIST status	None	None	None

6.8 AFE Configuration Functions

Table 14. AFE Configuration Functions

Function Name	Description	Parameters	Returns	Postcondition
CONFIG_P_GAIN(x)	Configure P_GAIN_SELECT register	x - Value to be written to P_GAIN_SELECT register.	None	None
CONFIG_T_GAIN(x)	Configure T_GAIN_SELECT register	x - Value to be written to T_GAIN_SELECT register.	None	None

6.9 Diagnostic Interface Functions

Table 15. Diagnostic Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
GET_PSMON()	Get the values of PSMON register. Note all 10 bits (PSMON1 + PSMON2 bits) are read.	None	None	None
GET_AFEDIAG()	Get the values of AFEDIAG register.	None	None	None
AFE_CFG_CONFIG()	Configure AFE_CFG register	x - Value to be written to AFE_CFG register.	None	None
AFEDIAG_CFG_CONFIG()	Configure AFEDIAG_CFG register	x - Value to be written to AFEDIAG_CFG register.	None	None

6.10 Digital Interface Functions

This section describes API functions of digital interface.

Table 16. Digital Interface Functions

Function Name	Description	Parameters	Returns	Postcondition
void DIGITAL_Interface_Config (UC diconfigValue)	Configure digital interface for communication	diconfigValue Data to be written to digital interface config register.	None	None

6.11 EEPROM Functions

This section describes API functions of EEPROM interface.

Table 17. EEPROM Functions

Function Name	Description	Parameters	Returns	Postcondition
void EEPROM_Read_Coeff (void)	Read coefficients from EEPROM	None	None	None
UC EEPROM_Read (UC numBytes, UC eepromOffset, UC * destPtr)	Read values from EEPROM	numBytes - Number of bytes to be read. eepromOffset - Offset of EEPROM valid range 0 to 127. destPtr - Pointing to the address where read data to be stored.	1 if successful 0 if unsuccessful	None
UC EEPROM_Write (UC pageNum, UC * sourcePtr)	Write EEPROM	pageNum - Page number of EEPROM - valid value 0 to 15. sourcePtr - Pointing to the data to be written to EEPROM.	1 if successful 0 if unsuccessful	None
void EEPROM_Read_Disabled (UC erasePagenum, UC * srcPtr, UC * dstPtr)	Read EEPROM location when EEPROM erase and program is in progress.	pageNum - Page number of EEPROM- valid value 0 to 15. srcPtr - Pointing to the data to be written to EEPROM. dstPtr - Pointing to the address where read data to be stored.	None	destination buffer shall have total 8- 0x00 values indicates that read to EEPROM while EEPROM erase/program is in progress, is disabled.
void EEPROM_Reset_Init (void)	Initializes all EEPROM related global variables	None	None	None

6.12 GPIO Functions

This section describes API functions of GPIO interface.

Table 18. GPIO Functions

Function Name	Description	Parameters	Returns	Postcondition
void GPIO_Config (UC gpioDir)	Configure GPIO	gpioDir - GPIO direction input or output	None	None
GET_GPIO1	Get GPIO1 input value	None	None	None
GET_GPIO2	Get GPIO2 input value	None	None	None
GPIO1_HIGH()	GPIO1 output is high	None	None	None
GPIO1_LOW()	GPIO1 output is low	None	None	None
GPIO2_HIGH()	GPIO2 output is high	None	None	None
GPIO2_LOW()	GPIO2 output is low	None	None	None
GPIO1_TOGGLE()	GPIO1 output is toggled	None	None	None
GPIO2_TOGGLE()	GPIO2 output is toggled	None	None	None

6.13 ISR Functions

This section describes API functions of interrupt service routine.

Table 19. ISR Functions

Function Name	Description	Parameters	Returns	Postcondition
void Interrupt_Reset_Init (void)	Initialization of PGA900 interrupts	None	None	None

6.14 Multiplexer Functions

This section describes API functions of analog and digital multiplexer functions.

Table 20. Multiplexer Functions

Function Name	Description	Parameters	Returns	Postcondition
AMUX_ACT_CONFIG (x)	Enable/disable analog input and/or output test multiplexer	x - value 0x00 to 0x02	None	None
AMUX_CONFIG (x)	Analog multiplexer control	x - value 0x00 to 0x0F	None	None
AMUX_TIN_MUX_CONFIG (x)	Multiplexer select for analog test multiplexer input	x	None	None
AMUX_TOUT_MUX_CONFIG (x)	Multiplexer select for analog test multiplexer output	x	None	None
TEMP_MUX_CONFIG (x)	Configure Temperature control register	x	None	None
TONDIG_MUX_CONFIG (x)	Configure the signals driven to the TONDIG test output	x - Value ranging from 0x00 to 0x34	None	None
TOPDIG_MUX_CONFIG (x)	Configure the signals driven to the TOPDIG test output	x - Value ranging from 0x00 to 0x34	None	None

6.15 OTP Functions

This section describes API functions of OTP interface.

Table 21. OTP Functions

Function Name	Description	Parameters	Returns	Postcondition
void OTP_Checksum (void)	Computes OTP Checksum	None	None	None

6.16 OWI Functions

This section describes API functions of one-wire interface.

Table 22. OWI Functions

Function Name	Description	Parameters	Returns	Postcondition
void OWI_Config (UC owiInt)	Configure OWI interface interrupt registers and Enable OWI clock	owiInt Data to be written to OWI_INTERRUPT_ENABLE register	None	None
OWI_TRANSCEIVER_DISABLE()	Disable OWI transceiver. OWI transceiver is disconnected from VDD.	None	None	None
OWI_CLOCK_ENABLE()	Enables OWI clock	None	None	None
void OWI_Activation_Handler (void)	Interrupt subroutine for OWI activation	None	None	None

6.17 PWM Functions

This section describes API functions of PWM interface.

Table 23. PWM Functions

Function Name	Description	Parameters	Returns	Postcondition
void PWM_On_Off_Cfg (US pwmOn, US pwmOff)	Configure PWM on and off time	pwmOn PWM on time pwmOff PWM off time	None	None
PWM_ENABLE()	Enable PWM block	None	None	None
PWM_DISABLE()	Disable PWM block	None	None	None

6.18 REMAP Functions

PGA900 has 8KB of Development RAM. The PGA900 device can run the code from Development RAM by configuring REMAP bit in the REMAP register.

Table 24. REMAP Functions

Function Name	Description	Parameters	Returns	Postcondition
void REMAP_Config (UC rValue)	Remaps the OTP memory to development RAM area and vice versa.	rValue Data to be written to REMAP register. Valid value 0x00 or 0x01	None	None

NOTE: Copy executable code from OTP to development RAM before setting REMAP bit to 1. Trace FIFO should not be used when code is running from development RAM.

6.19 Sys Tick Timer Functions

This section describes API functions of M0 system tick timer.

Table 25. Sys Tick Timer Functions

Function Name	Description	Parameters	Returns	Postcondition
SYST_Config (UL reloadCnt, UL systControl)	Configure Cortex M0 system timer	reloadCnt SysTick reload value systControl SysTick control value	None	None
SYST_Handler (void)	Interrupt subroutine for Cortex M0 system timer	None	None	None
SYST_GET_CVR()	Get SysTick current value	None	None	None

6.20 Trace FIFO Functions

This section describes API functions of trace FIFO interface.

Table 26. Trace FIFO Functions

Function Name	Description	Parameters	Returns	Postcondition
TRACE_FIFO_CONFIG (x)	Configure trace FIFO control and status register	x	None	None

NOTE: Trace FIFO should not be used when code is running from development RAM.

6.21 UART Functions

This section describes API functions of UART interface.

Table 27. UART Functions

Function Name	Description	Parameters	Returns	Postcondition
void UART_Config (UC uconfigValue, UC baudValue)	Configure UART	uconfigValue Data to be written to UART_CONFIG register. baudValue Data to be written to BAUD_RATE register.	None	None
void UART_Handler (void)	Interrupt subroutine for UART	None	None	None
void UART_Reset_Init (void)	Initializes all UART related global variables	None	None	None
SPI_PIN_MUX_SELECT ( )[1]	UART does not use SPI interface MOSI and SOMI pins	None	None	None
UART_PIN_MUX_SELECT ( )	UART uses the SPI interface MOSI and SOMI pins. (Also see *SPI_PIN_MUX_SELECT()*)	None	None	None
UART_ENABLE ( )	Enable UART block	None	None	None
UART_RX_INT_ENABLE ( )	Enable UART receive ready interrupt	None	None	None
UART_TX_INT_ENABLE ( )	Enable UART transmit empty interrupt	None	None	None

(1) When communicating on SPI, PIN_MUX has to be 0x00, that is, MOSI and SOMI pins are used by SPI (not UART).

6.22 WDT Functions

This section describes API functions of watch dog timer interface.

Table 28. WDT Functions

Function Name	Description	Parameters	Returns	Postcondition
WDT_ENABLE()	Enable watch dog timer	None	None	None
WDT_DISABLE()	Disable watch dog timer	None	None	None
WDT_TRIGGER(x)[1]	Load watch dog trigger value into WDOG_TRIG register	x Data to be written to WDOG_TRIG register.	None	None
WDT_RESET_CLEAR()	Clear watch dog reset bit by writing 1	None	None	None
WDT_TIMED_OUT()	Provides status of watchdog timed out	None	1 if watchdog is timed out	None

(1) Trace FIFO should not be used when code is running from development RAM.

6.23 Main Function

This section describes API function of main source file.

Main function contains loop. Main loop calls pressure and temperature coefficient calculation function, COMBUF command processing function and UART data transmit function.

Pressure and temperature coefficients are calculated every 2.048 ms.

COMBUF commands are processed only when COMBUF interface receives data. In the current implementation, if the COMBUF interface receives a value from 1 to 16 (inclusive), then it is treated as a command and the appropriate command processing function gets called. Find the supported commands with their functionality details in CMD_Reset_Init() function.

To show the functionality of the UART data receive and transmit feature, UART is configured as 9600 8-N-1, that is, baud rate 9600, data bits 8, parity none, and stop bit 1. At power on, the UART configuration is performed and its receive interrupt is enabled. Whenever the microcontroller receives data on UART, the main loop performs bitwise NOT operation on received data and transmit complementary data on UART.

Table 29. Main Function

Function Name	Description	Parameters	Returns	Postcondition
void main (void)	Main routine	None	None	None

6.24 Filter Functions

This section describes API functions of FIR filter.

Table 30. Filter Functions

Function Name	Description	Parameters	Returns	Postcondition
void FIR_Reset_Init (void)	Initializes all FIR filter related global variables	None	None	None
void Pchannel_Filter_Acc (void)	Accumulation of pressure channel ADC values	None	None	None
void Pchannel_FIR_Filter (void)	48-sample FIR filter	None	None	None
void Tchannel_Filter_Acc (void)	Accumulation of temperature channel ADC values	None	None	None
void Tchannel_FIR_Filter (void)	192-sample FIR filter	None	None	None

6.25 Application Functions

This section describes application specific functions.

Table 31. Application Functions

Function Name	Description	Parameters	Returns	Postcondition
void APP_Reset_Init(void)	Initializes application-specific global variables	None	None	None
void APP_Calculate_Coeff (void)	State machine for T coefficient and P calculation	None	None	None
void CFG_Peripheral_Config (void)	Configuration of PGA900 peripherals	None	None	None
void CMD_Reset_Init(void)	Initializes application-specific global variables	None	None	None
void CMD_Process(UC cmdNum)	Process commands. Total 16 commands are supported. List of the supported commands are available in CMD_Reset_Init() function.	cmdNum Command Number	None	None

Revision History

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		