PGA970 Software User's Guide

User's Guide

Literature Number: SLDU024
November 2016

Contents

Introduction

Texas instruments PGA970 is an interface device for LVDT sensors. The device incorporates analog front end that directly connects to the sense element and has voltage regulators and oscillator. The device also includes ADC, ARM Cortex-M0 microprocessor and FRAM memory. Sensor compensation algorithms can be implemented in software. The PGA970 includes multiple output interfaces.

This document describes PGA970 software architecture, source code directory structure and software driver details.

Software Architecture Overview

This section describes PGA970 software architecture.

Figure 2-1 shows PGA970 software architecture.

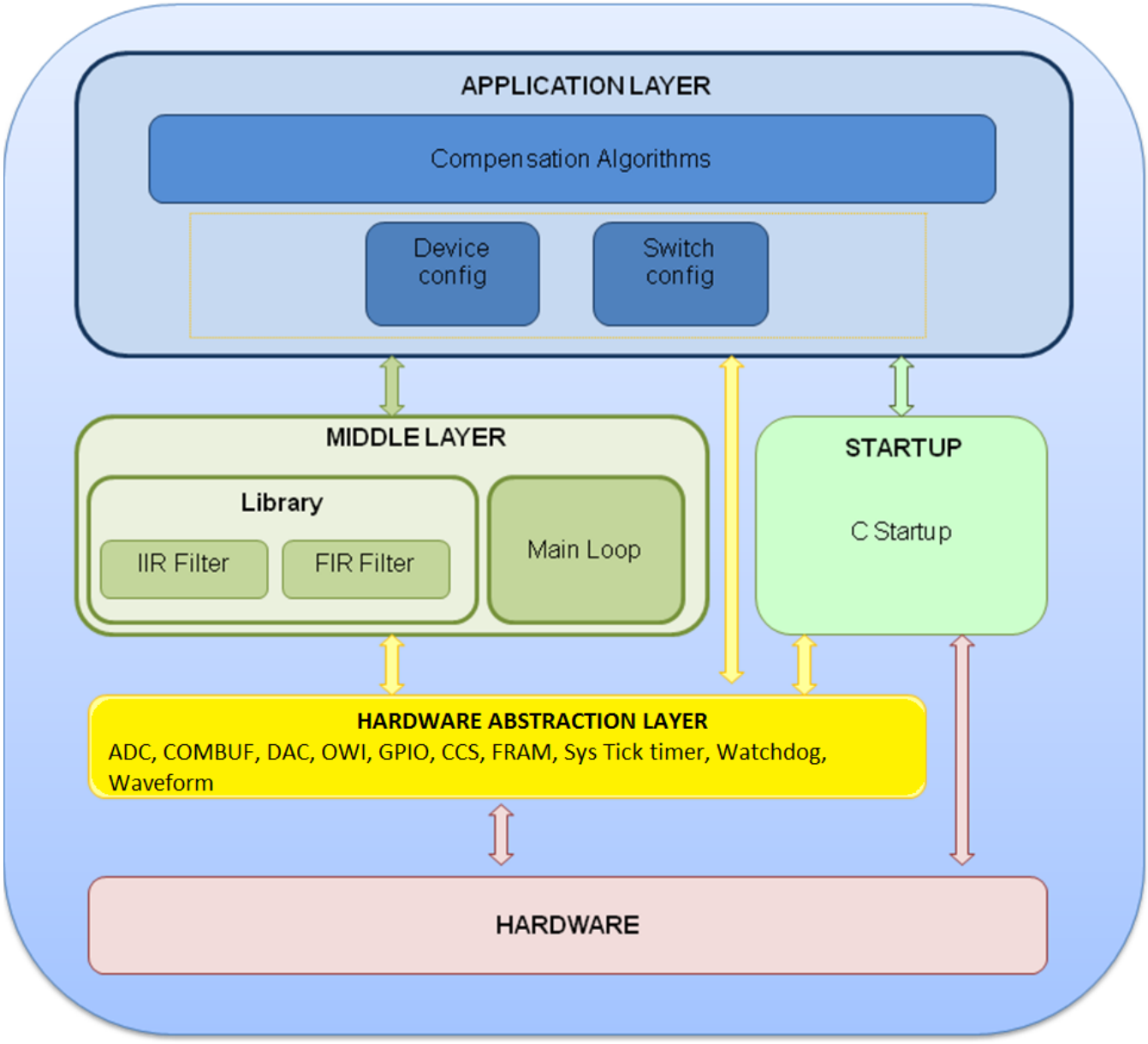

Figure 2-1. PGA970 Software Architecture

The PGA970 software consists of three layers, which are described below:

1. **Hardware Abstraction Layer (HAL):** The Hardware Abstraction Layer implements software that manipulates various PGA970 registers. The HAL register variables are in implemented in APIs (Application Program Interfaces) that the Application Layer uses to access the PGA970 Control and Status Registers.
2. **Middle Layer (ML):** The Middle Layer implements hardware-independent functionality including Filter routines, and C main function. The reference code includes example FIR filter routines.
3. **Application Layer (APP):** The Application Layer not only configures the PGA970 for specific user configuration but also implements the compensation algorithms. The reference code includes sample application which demonstrates the use of peripheral drivers.

In addition, the PGA970 software also contains power up routines that implement the following functions:

- Locate Vector Table
- Clear RAM

Source Code Directory Structure

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

Figure 3-1 shows PGA970 source code directory structure.

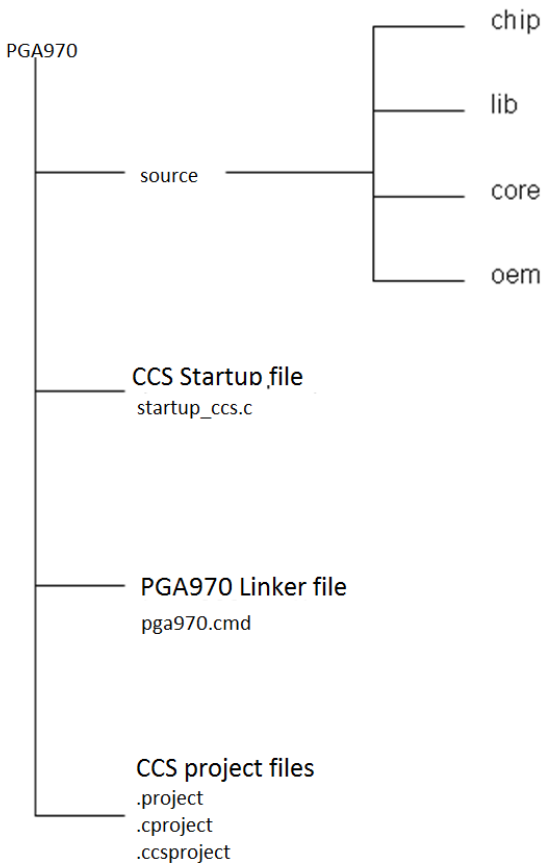

Figure 3-1. Source Code Directory Structure

- Source folder contains source files. Source files are arranged as per PGA970 software architecture.
  - The chip folder contains definition of hardware registers, initialization and configuration of PGA970 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 uses low level and middle layer functions.
- CCS startup file - Startup code for use with TI's Code Composer Studio.
- PGA970 linker file – pga970.cmd is a linker command file. The MEMORY directive defines the PGA970 memory configuration. The SECTIONS directive controls how sections are built and allocated.

3.1 List of Source Code Files

3.1.1 CHIP

The chip folder contains header files and source code files. Figure 3-2 is the snapshot of the chip folder.

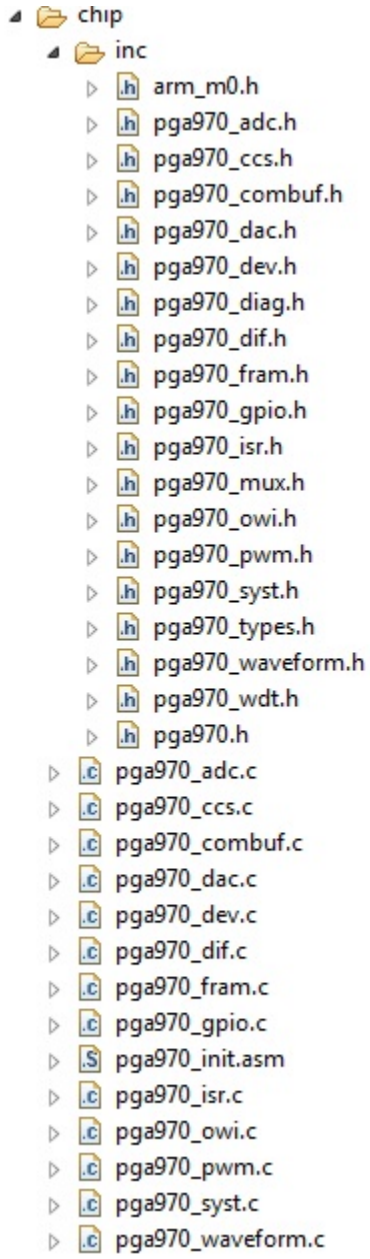

Figure 3-2. Source Code of CHIP Folder

Table 3-1. List of Source Files of CHIP Folder

SOURCE FILES	DESCRIPTION
pga970_adc.c and pga970_adc.h	Source file for analog to digital converter interface.
pga970_ccs.c and pga970_ccs.h	Source file for CCS module
pga970_combuf.c and combuf.h	Source file for COMBUF interface.
pga970_dac.c and pga970_dac.h	Source file for digital to analog converter interface.
pga970_dev.c and pga970_dev.h	Source file for PGA970 device.
pga970_dif.c and pga970_dif.h	Source file for digital interface.
pga970_fram.c and pga970_fram.h	Source files for FRAM
pga970_gpio.c and pga970_gpio.h	Source file for GPIO interface.
pga970_isr.c and pga970_isr.h	Source file for interrupt sub routine.
pga970_owi.c and pga970_owi.h	Source file for OWI interface
pga970_syst.c and pga970_syst.h	Source file for sys tick timer interface.
Pga970_pwm.h and pga970_pwm.c	Source file for PWM.
pga970_waveform.c and pga970_waveform.h	Source file for Waveform interface.
pga970.h	Definition of PGA970 control and status register.
pga970_mux.h	Source file for analog and digital multiplexer selection.
pga970_wdt.h	Source file for Watchdog interface.
pga970_types.h	Data types are type defined.
arm_m0.h	ARM Cortex M0 register definitions.
pga970_diag.h	Diagnostic bits are defined. There are three diagnostic registers PSMON1, PSMON2 and AFEDIAG.
pga970_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 • DATA, Development and Waveform Ram Test

3.1.2 LIB

The lib folder contains library header and source files. Figure 3-3 is the snap-shot of the lib folder.

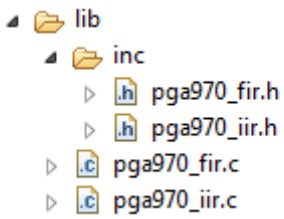

Figure 3-3. Source Code of LIB Folder

Table 3-2. List of Source Files of LIB Folder

SOURCE FILES	DESCRIPTION
pga970_fir.c and pga970_fir.h	Source files for FIR filter
pga970_iir.c and pga970_iir.c	Source files for IIR filter

3.1.3 CORE

The core folder contains main header file and source file. Figure 3-4 is the snap-shot of the core folder.

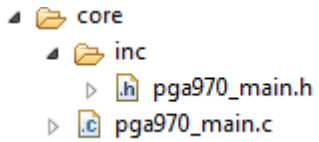

Figure 3-4. Source Code of CORE Folder

Table 3-3. List of Source Files of CORE Folder

SOURCE FILES	DESCRIPTION
pga970_main.c and pga970_main.h	Source files for main function

3.1.4 OEM

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

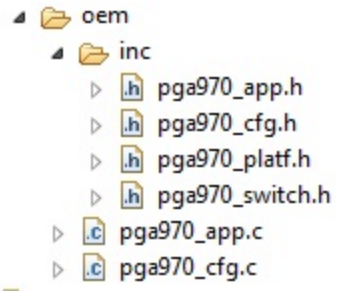

Figure 3-5. Source Code of OEM Folder

Table 3-4. List of Source Files of OEM Folder

SOURCE FILES	DESCRIPTION
pga970_app.c and pga970_app.h	Source files for application - Compensation algorithms are implemented.
pga970_cfg.c and pga970_cfg.h	Source files to configure PGA970 peripherals.
pga970_platf.h	All header files are included.
pga970_switch.h	Definition of compile time switches.

Software Configurations

This section describes configuration of PGA970 software.

4.1 Device Configuration

PGA970 peripherals can be configured using pga970_cfg.c source file. The configuration involves writing to specific values to registers.

The Device Configuration file, pga970_cfg.c, contains CFG_Peripheral_Config () function. This function can be modified to configure PGA970 peripherals.

Table 4-1 shows the list of PGA970 peripheral configuration functions.

Table 4-1. List of PGA970 Peripheral Configuration Functions

DEVICE MODULE	CONFIGURATION FUNCTION NAME	DESCRIPTION
ADC	ADC_Config() S1_CONFIG() S2_CONFIG() S1_S2_CONFIG() S1_S2_CONFIG() S3_CFG_1_CONFIG() DEMOD1_BPF_config() DEMOD1_LPF_config() DEMOD2_BPF_config() DEMOD2_LPF_config() ADC3_VI_LPF_Config() ADC3_VI_LPF_Config() ADC3_PTAT_LPF_Config()	Configuration of ADC.
COMBUF	COMBUF_RX_INT_ENABLE()	Enable COMBUF receive interrupt.
DAC	DAC_Config() DAC_Loopback_Config()	Configure DAC device module.
PGA970 Device Clock	M0_ConfigClock()	Configure PGA970 clock.
Digital Interface	DIGITAL_Interface_Config()	Configure digital interfaces (SPI and OWI).
GPIO	GPIO_Config()	Configure GPIO direction.
LED	LED_Config()	Configures LED.
ISR	NA	This module does not required configuration function.
Digital multiplexer	TOPDIG_MUX_CONFIG()	Configure digital multiplexer.
Analog multiplexer	AMUX_TIN_MUX_CONFIG() AMUX_TOUT_MUX_CONFIG()	Configure analog multiplexer.
FRAM	NA	This module does not required configuration function.
OWI	OWI_Config()	Configure one wire interface.

Table 4-1. List of PGA970 Peripheral Configuration Functions (continued)

DEVICE MODULE	CONFIGURATION FUNCTION NAME	DESCRIPTION
PWM	PWM_Config() PWM_Ch1_Config() PWM_Ch2_Config() PWM_Ch34_Config() PWM_Ch34_Output_Select()	Configures PWM
System Tick Timer	SYST_Config()	Configure system tick timer.
Watchdog Timer	NA	This module does not required configuration function.

Refer to Chapter 9 for more information.

Refer the *Functional Block Diagram* section of *PGA970 LVDT Sensor Signal Conditioner* (SLDS201) to get more information of PGA970 peripherals.

4.2 Conditional Compilation Switches

The user can compile in or compile out PGA970 peripheral functionality by configuring various compile time switches available in pga970_switch.h file. Compiling in the only-needed functions will allow code size optimization.

Table 4-2 shows list of conditional compiler switches.

Table 4-2. List of Conditional Compiler Switches

COMPILE SWITCH	DESCRIPTION
SYST_TESTING	When SYST_TESTING compile switch is 1 then source code of system timer SysTick module available into pga970_syst.c and pga970_syst.h gets compiled. When SYST_TESTING compile switch is 0 then source code of system timer SysTick module available into pga970_syst.c and pga970_syst.h gets suppressed.
DIAGNOSTIC_TESTING	When DIAGNOSTIC _TESTING compile switch is 1 then source code of diagnostic module available into pga970_diag.h gets compiled. When DIAGNOSTIC _TESTING compile switch is 0 then source code of diagnostic module available into pga970_diag.h gets suppressed.
GPIO_TESTING	When GPIO_TESTING switch is 1 then source code of GPIO module available into pga970_gpio.c and pga970_gpio.h gets compiled. When GPIO_TESTING switch is 0 then source code of GPIO module available into pga970_gpio.c and pga970_gpio.h gets suppressed.
FRAM_DATA_TESTING	When FRAM_DATA_TESTING compile switch is 1 then source code of FRAM Data module available into pga970_fram.c and pga970_fram.h gets compiled. When FRAM_DATA_TESTING switch is 0 then source code of FRAM Data module available into pga970_fram.c and pga970_fram.h gets suppressed.
FRAM_TEXT_CHECKSUM_CALC	When FRAM_TEXT_CHECKSUM_CALC switch is 1 then source code of checksum calculation for FRAM text area available into pga970_fram.c and pga970_fram.h gets compiled. When FRAM_TEXT_CHECKSUM_CALC switch is 0 then source code of checksum calculation for FRAM text area available into pga970_fram.c and pga970_fram.h gets suppressed.

Table 4-2. List of Conditional Compiler Switches (continued)

COMPILE SWITCH	DESCRIPTION
FRAM_DATA_CHECKSUM_CALC	When FRAM_DATA_CHECKSUM_CALC switch is 1 then source code of checksum calculation for FRAM data area available into pga970_fram.c and pga970_fram.h gets compiled. When FRAM_DATA_CHECKSUM_CALC switch is 0 then source code of checksum calculation for FRAM data area available into pga970_fram.c and pga970_fram.h gets suppressed.
WAVEFORM_TESTING	When WAVEFORM_TESTING switch is 1 then source code of waveform module available into pga970_waveform.c and pga970_waveform.h gets compiled. When WAVEFORM_TESTING switch is 0 then source code of waveform module available into pga970_waveform.c and pga970_waveform.h gets suppressed.
OWI_TESTING	When OWI_TESTING switch is 1 then source code of OWI module available into pga970_owi.c and pga970_owi.h gets compiled. When OWI_TESTING switch is 0 then source code of OWI module available into pga970_owi.c and pga970_owi.h gets suppressed.
LED_TESTING	When LED_TESTING switch is 1 then source code of LED module available into pga970_gpio.c and pga970_gpio.h gets compiled. When LED_TESTING switch is 0 then source code of LED module available into pga970_gpio.c and pga970_gpio.h gets suppressed.
PWM_TESTING	When PWM_TESTING switch is 1 then source code of PWM module available into pga970_pwm.c and pga970_pwm.h gets compiled. When PWM_TESTING switch is 0 then source code of PWM module available into pga970_pwm.c and pga970_pwm.h gets suppressed.
DEVRAM_TESTING	When DEVRAM_TESTING switch is 1 then source code of development RAM module available into pga970_cfg.c gets compiled. When DEVRAM_TESTING switch is 0 then source code of development RAM module available into pga970_cfg.c gets suppressed.
COMBUF_TESTING	When COMBUF_TESTING switch is 1 then source code of COMBUF module available into pga970_conbuf.c and pga970_combuf.h gets compiled. When COMBUF_TESTING switch is 0 then source code of COMBUF module available into pga970_conbuf.c and pga970_combuf.h gets suppressed.
CCS_TESTING	When CCS_TESTING switch is 1 then source code of CCS module available into pga970_ccs.c and pga970_ccs.h gets compiled. When CCS_TESTING switch is 0 then source code of CCS module available into pga970_ccs.c and pga970_ccs.h gets suppressed.
WATCH_DOG_TIMER_TESTING	When WATCH_DOG_TIMER_TESTING switch is 1 then source code of watch dog module available into pga970_wdt.h gets compiled. When WATCH_DOG_TIMER_TESTING switch is 0 then source code of watch dog module available into pga970_wdt.h gets suppressed.

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

Usage Example:

GPIO_TESTING compile time switch is set to 1 using following syntax #define GPIO_TESTING ON

GPIO_TESTING compile time switch is set to 0 using following syntax #define GPIO_TESTING OFF

By default all compile time switches are disabled.

4.3 Linker Configuration

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

The pga970.cmd is a CCS linker configuration file for PGA970. The pga970.cmd uses two directives: MEMORY and SECTIONS.

4.3.1 MEMORY

The MEMORY directive allows specifying a model of target memory. PGA970 contains 8KBytes of FRAM memory, 2KBytes of data SRAM, 256 Bytes of Waveform RAM and 8KBytes of Development RAM.

Figure 4-1 shows memory map of PGA970.

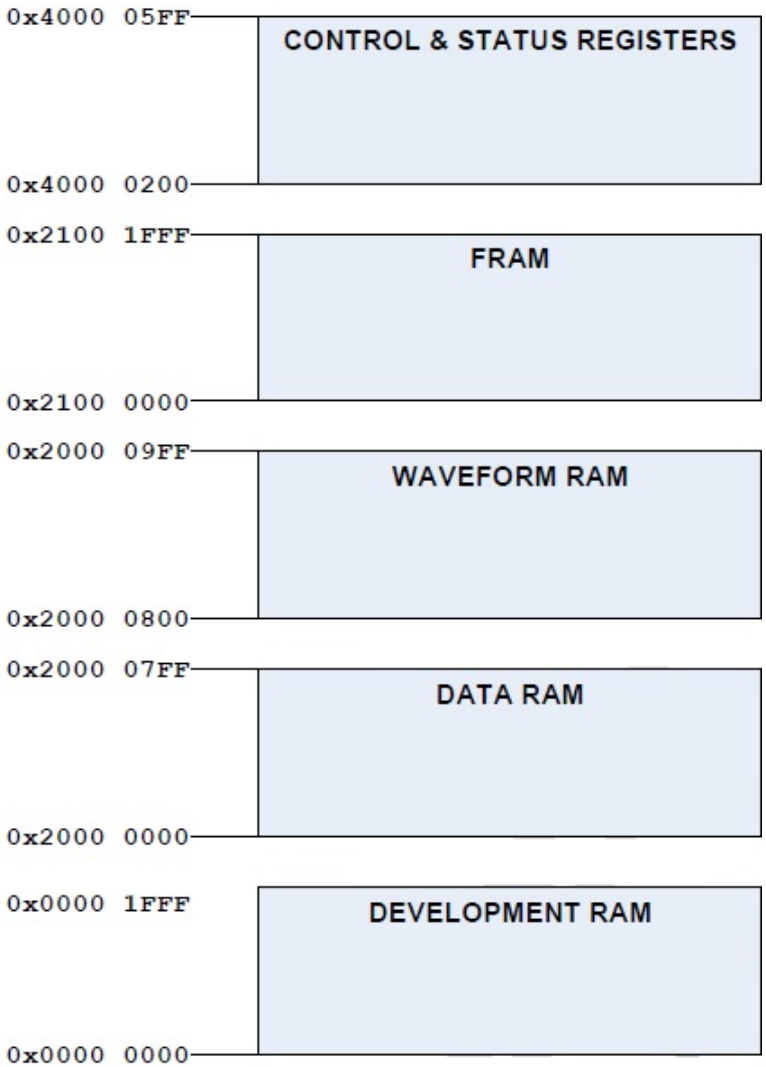

Figure 4-1. Memory Map of PGA970

4.3.2 SECTIONS

The SECTIONS directive controls how sections are built and allocated.

The SECTIONS directive controls how sections are built and allocated.

- Initialized sections: .text, .const, .cinit etc.
- Uninitialized sections: .bss, .stack, .sysmem etc.

Using SECTIONS directive, place these generated sections into memory. Information about MEMORY and SECTIONS directive can be found in *ARM Assembly Language Tools v16.9.0.LTS* (SPNU118).

Below is the code from pga970.cmd file.

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

/* System memory map */
MEMORY
{
    /* Application stored in and executes from internal flash
    Size of  FRAM is 6 KB */
    FRAM_TEXT (RX) : origin = APP_BASE, length = 0x00001800
    FRAM_DATA (RX) : origin = 0x00001800, length = 0x00000800
    /* Application uses internal RAM for data
    Size of SRAM is 2KB */
    SRAM (RWX) : origin = RAM_BASE, length = 0x00000800
    /* Application can be stored in DRAM and executes from DRAM
     Size of DRAM is 6KB */
    DRAM (RX) : origin = 0x21000000, length = 0x00001800
    /* Waveform RAM is 512 bytes */
    WRAM (RX) : origin = 0x20000800, length = 0x00000200
}

/* Section allocation in memory */
SECTIONS
{
    .intvecs:   > APP_BASE
    .text   :   > FRAM_TEXT
    .const  :   > FRAM_TEXT
    .cinit  :   > FRAM_TEXT
    .pinit  :   > FRAM_TEXT
    .init_array : > FRAM_TEXT
    .FRAM_TEXT_checksum   : > 0x000017FC
    .waveform_table       : > WRAM

    .vtable :   > RAM_BASE
    .data   :   > SRAM
    .bss    :   > SRAM
    .sysmem :   > SRAM
    .stack  :   > 0x20000000
}
```

PGA970 device contains 8KBytes of FRAM and Development RAM. Certain section can be configured to run out of Development RAM. Only 6KB of FRAM and Development RAM can be used for code rest of the area is reserved for User defined DATA and named as FRAM_DATA area.

Following example shows how to run certain section out of Development RAM:

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

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

In above example CODE_SECTION pragma allocates space for the symbol ADC_Handler.

- Modify SECTIONS part of the pga970.cmd file as follows:

```
/* Section allocation in memory */
SECTIONS
{
         intvecs:   > APP_BASE
          .text   :   > FRAM_TEXT
          .const  :   > FRAM_TEXT
          .cinit  :   > FRAM_TEXT
          .pinit  :   > FRAM_TEXT
          .init_array : > FRAM_TEXT
          .FRAM_TEXT_checksum   : > 0x000017FC
          .waveform_table       : > WRAM

          .vtable :   > RAM_BASE
          .data   :   > SRAM
          .bss    :   > SRAM
          .sysmem :   > SRAM
          .stack  :   > 0x20000000
   .ram_exec_sect : load = FRAM_TEXT, run = DRAM, table(BINIT)
}
```

The last statement in the SECTIONS part indicates that, ram_exec_sect section is loaded into FRAM_TEXT and runs from DRAM. That is ADC_Handler function is loaded into FRAM_TEXT and run out of Development RAM.

Software Development Tools and Build Environment

This section describes software development tools version and compiler option used for PGA970 code development.

- For PGA970 software development, Code Composer Studio Version: 5.5.0.00077 is used.
- The current version of the PGA970 code is built with ARM TI v5.1.1 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 PGA970 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
```

Information about compile switches can be found in *ARM Optimizing C/C++ Compiler v16.9.0.LTS* (SPNU151).

For PGA970 software development, the latest version of Code Composer Studio may be used.

- Download Code Composer Studio installation: http://processors.wiki.ti.com/index.php/Download_CCS
- If you would like extended M0 simulation features, please use version 5.5.0

Application Software

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

6.1 Sample Application Software

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

6.1.1 Compensation Equation

The second order TC compensation equation is as follows:

$$DAC = (h_0 + h_1TADC + h_2TADC^2) + (g_0 + g_1TADC + g_2TADC^2)PADC + (n_0 + n_1TADC + n_2TADC^2)PADC^2$$

where

- P is PADC Value
- T is TADC Value

(1)

6.1.2 Normalization of ADC Values

Normalize as follows:

- $P_n = PADC/2^{14}$
- $T_n = TADC/2^{14}$
- $D_n = DAC/2^{14}$

Based on these normalized values, the compensation algorithm can be written as:

$$D_n = (h_{0n} + h_{1n}T_n + h_{2n}T_n^2) + (g_{0n} + g_{1n}T_n + g_{2n}T_n^2)P_n + (n_{0n} + n_{1n}T_n + n_{2n}T_n^2)P_n^2 \quad (2)$$

6.1.3 Scaling of Coefficient

Store coefficients in FRAM DATA area by multiplying the floating point version by 2^{14}.

$h_{0EE} = round(h_{0n} \times 2^{14})$
$h_{1EE} = round(h_{1n} \times 2^{14})$
$h_{2EE} = round(h_{2n} \times 2^{14})$
$g_{0EE} = round(g_{0n} \times 2^{14})$
$g_{1EE} = round(g_{1n} \times 2^{14})$
$g_{2EE} = round(g_{2n} \times 2^{14})$
$n_{0EE} = round(n_{0n} \times 2^{14})$
$n_{1EE} = round(n_{1n} \times 2^{14})$
$n_{2EE} = round(n_{2n} \times 2^{14})$

Note that the normalization should be chosen such that each coefficient is 2 bytes in width in order to fit each coefficient in 2 bytes in the FRAM data area.

6.1.4 Algorithm and Implementation

6.1.4.1 Algorithm

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

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

$$DAC = D_n 2^{14} \quad (3)$$

6.1.4.2 Implementation

The pseudo-code for implementing the above algorithm is:

1. h = h2EE * TADC
2. h = h>>14
3. h = h + h1EE
4. h = h * TADC
5. h = h>>14
6. h = h + h0EE
7. g = g2EE * TADC
8. g = g>>14
9. g = g + g1EE
10. g = g * TADC
11. g = g>>14
12. g = g + g0EE
13. n = n2EE * TADC
14. n = n>>14
15. n = n + n1EE
16. n = n * TADC
17. n = n>>14
18. d = n * PADC
19. d = d>>14
20. d = d + g
21. d = d * PADC
22. d = d>>14
23. d = d + h
24. d = d

6.1.4.3 Calculation Timing

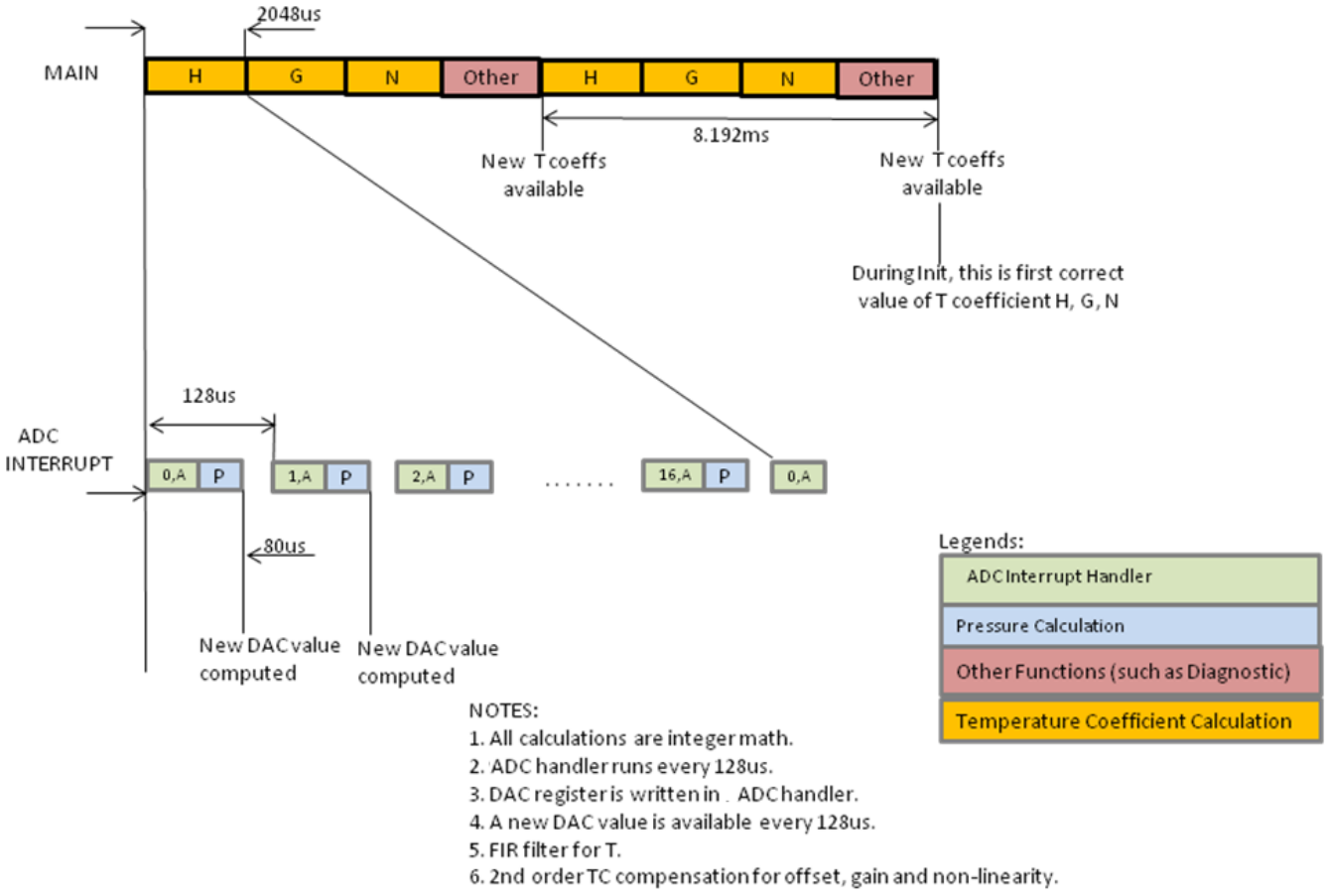

Figure 6-1. Calculation Timing

The P ADC is set up for 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 is read every 128 µs. The Free slots can be used to implement filters and diagnostics.

6.1.5 FRAM DATA Area

Table 6-1. FRAM DATA Address

Variables	FRAM Data Area Address	
	MSB	LSB
H0	0x00001801	0x00001800
G0	0x00001803	0x00001802
N0	0x00001805	0x00001804
H1	0x00001807	0x00001806
G1	0x00001809	0x00001808
N1	0x0000180B	0x0000180A
H2	0x0000180D	0x0000180C
G2	0x0000180F	0x0000180E
N2	0x00001811	0x00001810

NOTE: The size of coefficients is 16 bits.

Table 6-1 shows FRAM Data memory location assigned to the coefficient.

6.2 Building Application Software

User can build their application software using two approaches.

6.2.1 Addition of Application Functions

User can develop their application software by developing new functions or modifying the functions which are available into the reference sample application. The reference sample application is available into pga970_app.c and pga970_app.h. New functions can be added into existing sample application source files pga970_app.c and pga970_app.h.

Call new developed functions from APP_Calculate_Coeff() function available into pga970_app.c.

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

User can configure desired application specific peripheral of PGA970 by modifying CFG_Peripheral_Config() function available into source file pga970_cfg.c.

6.2.2 Addition of Application Source Files

User can add application specific new source files into oem folder. User need to create source file (.c file) and add into oem\ folder. Similarly header file need to be created (if required) and add into oem\inc folder. Include new header file into pga970_platf.h.

Call new developed functions from APP_Calculate_Coeff() function available into pga970_app.c.

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

Coding Standards

For PGA970 code development, standardize common code development practices are used.

Summary of coding standards rules are as below:

- 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 shall follow a loop or a conditional construct even if there is only one statement.
- Each source file and header file shall include a file header.
- Each function shall include a function header that shall state the purpose of the function and provide documentation for the function. This documentation includes: A full description of each parameter, the return type, all possible return values, required pre-conditions and/or defined post conditions, and a reference to related functions.

Example:

```
/*================================================================= */
/**
 * ADC_Config() Configure ADC.
 *
 * @param configValue Data to be written to ADC config register.
 *
 * @param gainValue Data to be written to  channel gain select register.
 *
 * @return none
 *
 * @post none
 *
 * @see ADC_Config.c
 */
/* ================================================================= */
```

- The file name of a source file shall indicate the function of the file. Example - pga970_adc.c
- All external API functions shall be prefixed with the module name followed by an underscore. Example - ADC_Config()
- All local functions shall be named consistently. The local functions naming convention should be different from the convention of global functions/external APIs.
  Example:
  ADC_Config() - global function
  process() - local function
- Variable names shall be consistent.
  Example:
  ADC_Count - global variable
  cmdNum - local variable
- There shall be three blank lines between the closing brace of one function and 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 shall name the header file holding the function declaration. This is to detect and correct errors in documentation linkage.

  Example:
  ADC_Config() function is properly documented in header file.

```
/* ========================================================================== */
/**
 * ADC_Config() Configure ADC channels.
 *
 * @param adc1configValue Data to be written to ADC1_CONFIG register.
 *
 * @param adc2configValue Data to be written to ADC2_CONFIG register.
 *
 * @param adc3configValue Data to be written to ADC3_CONFIG register.
 *
 * @param s1s2Cfg1 Data to be written to S1_S2_DEMOD_CFG_1 register.
 *
 * @param s3Cfg1 Data to be written to S3_ADC_CFG_1 register.
 *
 * @return none
 *
 * @post none
*/
/* ========================================================================== */
```

  Source file omit the remaining parameters.

```
/* ========================================================================== */
/**
 * @fn ADC_Config() Configure ADC channels.
 *
 * @see pga970_adc.h
 */
/* ========================================================================== */
void ADC_Config(UC demod1configValue, UC demod2configValue, UC adc3configValue,
                UC s1s2Cfg1, UC s3Cfg1)
{
   S1_S2_DEMOD_CFG_1 = s1s2Cfg1;
   S3_ADC_CFG_1 = s3Cfg1;
   /*configure the connection of S1, S2 and S3 output*/
   AMUX_CTRL = ((AMUX_CTRL & ~0x0E) |
                  ((S1_OP_TO_ADC1 | S2_OP_TO_ADC2 | S3_OP_TO_ADC3) & 0x0E));
   DEMOD1_CONFIG = demod1configValue;
   DEMOD2_CONFIG = demod2configValue;
   ADC3_CONFIG = adc3configValue;
}
```

Types

The following types are defined in the pga970_types.h.

8.1 SC

```
typedef signed char             SC;
```

and is used for the following purpose
SC – Signed 8-bit integer

8.2 UC

```
typedef unsigned char             UC;
```

and is used for the following purpose
UC – Unsigned 8-bit integer

8.3 S2

```
typedef signed short             S2;
```

and is used for the following purpose
S2 – Signed 16-bit integer

8.4 US

```
typedef unsigned short             US;
```

and is used for the following purpose
US – Unsigned 16-bit integer

8.5 SL

```
typedef signed long               SL;
```

and is used for the following purpose
SL – Signed 32-bit integer

8.6 UL

```
typedef unsigned long             UL;
```

and is used for the following purpose
UL – Unsigned 32-bit integer

8.7 SI

```
typedef signed int                SI;
```

and is used for the following purpose
SI – Signed 32-bit integer

8.8 UI

```
typedef unsigned int              UI;
```

and is used for the following purpose
UI – Unsigned 32-bit integer

8.9 FLOAT

```
typedef float                   FLOAT;
```

and is used for the following purpose
FLOAT – 32-bit integer

8.10 VSC

```
typedef volatile signed char      VSC;
```

and is used for the following purpose
VSC – Volatile signed 8-bit integer

8.11 VUC

```
typedef volatile unsigned char    VUC;
```

and is used for the following purpose
VUC – Volatile unsigned 8-bit integer

8.12 VS2

```
typedef volatile signed short     VS2;
```

and is used for the following purpose
VS2 – Volatile signed 16-bit integer

8.13 VUS

```
typedef volatile unsigned short   VUS;
```

and is used for the following purpose
VUS – Volatile unsigned 16-bit integer

8.14 VSL

```
typedef volatile signed long     VSL;
```

and is used for the following purpose
VSL – Volatile signed 32-bit integer

8.15 VUL

```
typedef volatile unsigned long     VUL;
```

and is used for the following purpose
VUL – Volatile unsigned 32-bit integer

8.16 VSI

```
typedef volatile signed int     VSI;
```

and is used for the following purpose
VSI – Volatile signed 32-bit integer

8.17 VUI

```
typedef volatile unsigned int     VUI;
```

and is used for the following purpose
VUI – Volatile unsigned 32-bit integer

8.18 VFLOAT

```
typedef volatile float     VFLOAT;
```

and is used for the following purpose
VFLOAT – Volatile 32-bit integer

API Functions

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

NOTE:

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

9.1 ADC Functions

This section describes API functions of ADC interface.

void ADC_Config (UC demod1configValue, UC demod2configValue, UC adc3configValue, UC s1s2Cfg1, UC s3Cfg1)
ADC_Config() – Configure ADC channels.

- Parameters:
  - demod1configValue Data to be written to ADC1_CONFIG register.
  - demod2configValue Data to be written to ADC2_CONFIG register.
  - adc3configValue Data to be written to ADC3_CONFIG register.
  - s1s2Cfg1 Data to be written to S1_S2_DEMOD_CFG_1 register.
  - s3Cfg1 Data to be written to S3_ADC_CFG_1 register.
- Returns:
  - None
- Post condition:
  - None

interrupt void ADC_Handler (void)
ADC_Handler() – Interrupt subroutine for ADC. Read T channel data into variable FIR_AccadcValue[1] and perform arithmetic calculation on DEMOD1 and DEMOD2 data.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void ADC_Reset_Init (void)
ADC_Reset_Init() – Initializes all ADC related global variables.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void DEMOD1_BPF_config(UL B1config, UL A2config, UL A3config)
DEMOD1_BPF_config – Initializes DEMOD1 BPF registers.

- Parameters:
  - B1config Configure B1 BPF Register of DEMOD1
  - A2config Configure A2 BPF Register of DEMOD1
  - A3config Configure A3 BPF Register of DEMOD1
- Returns:
  - None
- Post condition:
  - None

void DEMOD1_LPF_config(S2 B1config, S2 A2config)
DEMOD1_LPF_config – Initializes DEMOD1 LPF registers.

- Parameters:
  - B1config Configure B1 LPF Register of DEMOD1
  - A2config Configure A2 LPF Register of DEMOD1
- Returns:
  - None
- Post condition:
  - None

void DEMOD2_BPF_config(UL B1config, UL A2config, UL A3config)
DEMOD2_BPF_config – Initializes DEMOD2 BPF registers.

- Parameters:
  - B1config Configure B1 BPF Register of DEMOD2
  - A2config Configure A2 BPF Register of DEMOD2
  - A3config Configure A3 BPF Register of DEMOD2
- Returns:
  - None
- Post condition:
  - None

void DEMOD2_LPF_config(S2 B1config, S2 A2config)
DEMOD2_LPF_config – Initializes DEMOD2 LPF registers.

- Parameters:
  - B1config Configure B1 LPF Register of DEMOD2
  - A2config Configure A2 LPF Register of DEMOD2
- Returns:
  - None
- Post condition:
  - None

void ADC3_VBAT_LPF_Config (S2 B1config, S2 A2config)
ADC3_VBAT_LPF_config – Initializes VBAT LPF registers.

- Parameters:
  - B1config Configure B1 LPF Register of VBAT
  - A2config Configure A2 LPF Register of VBAT
- Returns:
  - None
- Post condition:
  - None

void ADC3_VI_LPF_Config (S2 B1config, S2 A2config)
ADC3_VI_LPF_config – Initializes VI LPF registers.

- Parameters:
  - B1config Configure B1 LPF Register of VI
  - A2config Configure A2 LPF Register of VI
- Returns:
  - None
- Post condition:
  - None

void ADC3_PTAT_LPF_Config (S2 B1config, S2 A2config)
ADC3_PTAT_LPF_config – Initializes PTAT LPF registers.

- Parameters:
  - B1config Configure B1 LPF Register of PTAT
  - A2config Configure A2 LPF Register of PTAT
- Returns:
  - None
- Post condition:
  - None

S1_CONFIG(x)
S1_CONFIG() – Configures ADC1 gain, SEM, Inversion.

- Parameters:
  - x value to be written in S1_CFGregister
- Returns:
  - None
- Post condition:
  - None

S2_CONFIG(x)
S2_CONFIG() – Configures ADC2 gain, SEM, Inversion.

- Parameters:
  - x value to be written in S2_CFG register
- Returns:
  - None
- Post condition:
  - None

S3_CONFIG(x)
S3_CONFIG() – Configures ADC3 gain, SEM, Inversion.

- Parameters:
  - x value to be written in S3_CFG register
- Returns:
  - None
- Post condition:
  - None

S1_S2_CONFIG(x)
S1_S2_CONFIG() – Configures S1 S2 common mode bias And VCM enable.

- Parameters:
  - x value to be written in S2_CFG register
- Returns:
  - None
- Post condition:
  - None

S3_CFG_1_CONFIG(x)
S3_CFG_1_CONFIG() – Configures S3 PTAT gain and S3 inversion.

- Parameters:
  - x value to be written in S3_CFG_1 register
- Returns:
  - None
- Post condition:
  - None

CONFIG_ALPWR(x)
CONFIG_ALPWR () – Configure ALPWR register.

- Parameters:
  - x value to be written to ALPWR register
- Returns:
  - None
- Post condition:
  - None

9.2 COMBUF Functions

This section describes API functions of COMBUF interface.

interrupt void COMBUF_Handler (void)
COMBUF_Handler() – Interrupt subroutine for COMBUF.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void COMBUF_Reset_Init (void)
COMBUF_Reset_Init() – Initializes all COMBUF related global variables.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

COMBUF_RX_INT_DISABLE ( )
COMBUF_RX_INT_DISABLE() – Disable COMBUF receive interrupt.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

COMBUF_RX_INT_ENABLE ( )
COMBUF_RX_INT_ENABLE() – Enable COMBUF receive interrupt.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

COMBUF_RX_STATUS_CLEAR ( )
COMBUF_RX_STATUS_CLEAR() – Clear COMBUF receive status bit by writting '1'.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.3 CCS Functions

This section describes API functions of COMBUF interface.

void CCS_Config (UC ccsconfigValue)
CCS_Config() – Configures CCS register.

- Parameters:
  - ccsconfigValue Configures CCS_CTRL register
- Returns:
  - None
- Post condition:
  - None

9.4 DAC Functions

This section describes API functions of DAC interface.

void DAC_Config (UC dacconfigValue, UC dacmodValue, UC dacgainValue)
DAC_Config() – Configure DAC.

- Parameters:
  - 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
- Returns:
  - None
- Post condition:
  - None

void DAC_Loopback_Config (UC lpconfigValue)
DAC_Loopback_Config() – Configure DAC loop-back.

- Parameters:
  - lpconfigValue Data to be written to loop-back control register
- Returns:
  - None
- Post condition:
  - None

void DAC_Reset_Init (void)
DAC_Reset_Init() – Initializes all DAC related global variables.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.5 Microcontroller Interface Functions

This section describes API functions of PGA970 device specific functions.

void M0_ConfigClock (UC value)
M0_ConfigClock() – Configure Cortex M0 clock.

- Parameters:
  - value Data to be written to clock control register
- Returns:
  - None
- Post condition:
  - None

MICRO_INTERFACE_CONFIG(x)
MICRO_INTERFACE_CONFIG() – Configure Micro interface control register.

- Parameters:
  - x value 0x00 to 0x07
- Returns:
  - None
- Post condition:
  - None

9.6 Digital Interface Functions

This section describes API functions of digital interface.

void DIGITAL_Interface_Config (UC diconfigValue)
DIGITAL_Interface_Config() – Configure digital interface for communication.

- Parameters:
  - diconfigValue Data to be written to digital interface config register
- Returns:
  - None
- Post condition:
  - None

9.7 Diagnostic Interface Functions

This section describes API functions of diagnostic interface.

GET_PSMON()
GET_PSMON() – Get the values of PSMON register. Please note all 10 bits (PSMON1+PSMON2 bits) are read.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_AFEDIAG()
GET_AFEDIAG() – Get the values of AFEDIAG register.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_AFEDIAG1()
GET_AFEDIAG1() – Get the values of AFEDIAG1 register.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_AFEDIAG3()
GET_AFEDIAG3() – Get the values of AFEDIAG3 register.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_AFEDIAG4()
GET_AFEDIAG4() – Get the values of AFEDIAG4 register.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

AFEDIAG_CFG_CONFIG()
AFEDIAG_CFG_CONFIG() – Configure AFEDIAG_CFG register.

- Parameters:
  - x Value to be written to AFEDIAG_CFG register
- Returns:
  - None
- Post condition:
  - None

9.8 FRAM Functions

This section describes API functions of FRAM interface.

void FRAM_DATA_Read_Coeff (void)
FRAM_DATA_Read_Coeff() – Read coefficients from FRAM Data area.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

UC FRAM_DATA_Read (UC numBytes, UC framdataOffset, void * destPtr)
FRAM_DATA_Read() – Read values from FRAM Data.

- Parameters:
  - numBytes Number of bytes to be read
  - framdataOffset Offset of FRAM DATA area valid range 0 to 2047
  - destPtr Pointing to the address where read data to be stored
- Returns:
  - 1 if success and 0 if unsuccessful
- Post condition:
  - None

UC FRAM_DATA_Write (UC numBytes, UC framdataOffset, void * srcPtr)
FRAM_DATA_Write() – Write to FRAM Data area.

- Parameters:
  - numBytes Number of Bytes to write
  - framdataOffset Offset of FRAM DATA area valid range 0 to 2047
  - sourcePtr Pointing to the data to be written to FRAM Data Area
- Returns:
  - 1 if success and 0 if unsuccessful
- Post condition:
  - None

void FRAM_TEXT_Checksum (void)
FRAM_TEXT_Checksum() – Calculates Checksum for FRAM Text area.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void FRAM_DATA_Checksum (void)
FRAM_DATA_Checksum() – Calculates Checksum for FRAM Data area.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.9 GPIO Functions

This section describes API functions of GPIO interface.

void GPIO_Config (UC gpioDir)
GPIO_Config() – Configure GPIO.

- Parameters:
  - gpioDir GPIO direction input or output
- Returns:
  - None
- Post condition:
  - None

GET_GPIO1
GET_GPIO1() – Get GPIO1 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_GPIO2
GET_GPIO2() – Get GPIO2 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_GPIO3
GET_GPIO3() – Get GPIO3 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_GPIO4
GET_GPIO4() – Get GPIO4 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_GPIO5
GET_GPIO5() – Get GPIO5 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GET_GPIO6
GET_GPIO6() – Get GPIO6 input value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO1_HIGH()
GPIO1_HIGH() – GPIO1 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO2_HIGH()
GPIO2_HIGH() – GPIO2 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO3_HIGH()
GPIO3_HIGH() – GPIO3 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO4_HIGH()
GPIO4_HIGH() – GPIO4 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO5_HIGH()
GPIO5_HIGH() – GPIO5 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO6_HIGH()
GPIO6_HIGH() – GPIO6 output is high.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO1_LOW()
GPIO1_LOW() – GPIO1 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO2_LOW()
GPIO2_LOW() – GPIO2 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO3_LOW()
GPIO3_LOW() – GPIO3 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO4_LOW()
GPIO4_LOW() – GPIO4 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO5_LOW()
GPIO5_LOW() – GPIO5 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO6_LOW()
GPIO6_LOW() – GPIO6 output is low.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO1_TOGGLE()
GPIO1_TOGGLE() – GPIO1 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO2_TOGGLE()
GPIO2_TOGGLE() – GPIO2 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO3_TOGGLE()
GPIO3_TOGGLE() – GPIO3 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO4_TOGGLE()
GPIO4_TOGGLE() – GPIO4 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO5_TOGGLE()
GPIO5_TOGGLE() – GPIO5 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

GPIO6_TOGGLE()
GPIO6_TOGGLE() – GPIO6 output is toggled.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void PIN_MUX_Config (UC pinMuxConfig)
PIN_MUX_Config() – Configure PIN_MUX register.

- Parameters:
  - pinMuxConfig Configures PIN_MUX register
- Returns:
  - None
- Post condition:
  - None

void LED_Config (UC LEDon)
LED_Config() – Configure LED.

- Parameters:
  - LEDon Configures LED
- Returns:
  - None
- Post condition:
  - None

9.10 ISR Functions

This section describes API functions of interrupt service routine.

void Interrupt_Reset_Init (void)
Interrupt_Reset_Init() – Initialization of PGA970 interrupts.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.11 Multiplexer Functions

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

AMUX_ACT_CONFIG (x)
AMUX_ACT_CONFIG() – Enable/disable analog input and/or output test multiplexer.

- Parameters:
  - x value 0x00 to 0x02
- Returns:
  - None
- Post condition:
  - None

AMUX_CTRL_CONFIG (x)
AMUX_CTRL_CONFIG() – Analog multiplexer control.

- Parameters:
  - x value 0x00 to 0xFF
- Returns:
  - None
- Post condition:
  - None

AMUX_TOUT_MUX_CONFIG (x)
AMUX_TOUT_MUX_CONFIG() – Multiplexer select for analog test multiplexer output.

- Parameters:
  - x
- Returns:
  - None
- Post condition:
  - None

AMUX_TIN_MUX_CONFIG (x)
AMUX_TIN_MUX_CONFIG() – Multiplexer select for analog test multiplexer input.

- Parameters:
  - x
- Returns:
  - None
- Post condition:
  - None

TOPDIG_MUX_CONFIG (x)
TOPDIG_MUX_CONFIG() – Configure the signals driven to the TOPDIG test output.

- Parameters:
  - x Value ranging from 0x00 to 0x34
- Returns:
  - None
- Post condition:
  - None

9.12 OWI Functions

This section describes API functions of one wire interface.

void OWI_Config (UC owiInt)
OWI_Config() – Configure OWI interface interrupt registers and Enable OWI clock.

- Parameters:
  - owiInt Data to be written to OWI_INTERRUPT_ENABLE register
- Returns:
  - None
- Post condition:
  - None

OWI_TRANSCEIVER_DISABLE()
OWI_TRANSCEIVER_DISABLE() – Disable OWI transceiver. OWI Transceiver is disconnected from VDD.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

OWI_CLOCK_ENABLE()
OWI_CLOCK_ENABLE() – Enables OWI clock.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void OWI_Activation_Handler (void)
OWI_Activation_Handler() – Interrupt subroutine for OWI activation.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.13 REMAP Functions

PGA970 has 6KBytes of Development RAM. The PGA970 device can run the code from Development RAM by configuring REMAP bit in the REMAP register.

void REMAP_Config (UC rValue)
REMAP_Config() – Remaps the FRAM memory to Development RAM area and vice versa.

- Parameters:
  - rValue Data to be written to REMAP register, Valid value 0x00 or 0x01
- Returns:
  - None
- Post condition:
  - None
- Attention:
  - Copy executable code from FRAM to Development RAM before setting REMAP bit to 1.

9.14 Sys Tick Timer Functions

This section describes API functions of M0 system tick timer.

SYST_Config (UL reloadCnt, UL systControl)
SYST_Config() – Configure Cortex M0 system timer.

- Parameters:
  - reloadCnt SysTick reload value
  - systControl SysTick control value
- Returns:
  - None
- Post condition:
  - None

SYST_Handler (void)
SYST_Handler() – Interrupt subroutine for Cortex M0 system timer.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

SYST_GET_CVR()
SYST_GET_CVR() – Get SysTick current value.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.15 Waveform Functions

This section describes API functions of waveform.

void WAVEFORM_Config(UC tableLength, US dacoffset, UC genCtrl)
WAVEFORM_Config() – Configure waveform generator registers.

- Parameters:
  - tablelength Waveform Table length
  - dacoffset Waveform bias value
  - genctrl Data to be written in Waveform control register
- Returns:
  - None
- Post condition:
  - None

LVDT_OP_CTRL_CONFIG(x)
LVDT_OP_CTRL_CONFIG() – Configure LVDT_OP_CTRL register.

- Parameters:
  - x Data to be written in LVDT_OP_CTRL register
- Returns:
  - None
- Post condition:
  - None

LVDT_LPBK_CTRL_CONFIG(x)
LVDT_LPBK_CTRL_CONFIG() – Configure LVDT_LPBK_CTRL register.

- Parameters:
  - x Data to be written in LVDT_LPBK_CTRL register
- Returns:
  - None
- Post condition:
  - None

9.16 PWM Functions

This section describes API functions of PWM interface.

PWM_Config(US timebase, US ADCsamplepoint)
PWM_Config() – Configures PWM time-base, ADC sample point and enables PWM.

- Parameters:
  - timebase Configures PWM_TIME_BASE register
  - ADCsamplepoint Configures PWM_ADC_SAMPLEPOINT register
- Returns:
  - None
- Post condition:
  - None

PWM_Config(US timebase, US ADCsamplepoint)
PWM_Config() – Configures PWM time-base, ADC sample point and enables PWM.

- Parameters:
  - timebase Configures PWM_TIME_BASE register
  - ADCsamplepoint Configures PWM_ADC_SAMPLEPOINT register
- Returns:
  - None
- Post condition:
  - None

PWM_Ch1_Config(US cmpOn, US maxOn, UC invertCh1)
PWM_Ch1_Config() – Configures PWM channel 1 registers.

- Parameters:
  - cmpOn Configures PWM_CH1_CMP_ON register
  - MaxOn Configures PWM_CH1_MAX_ON_TIME register
  - invertCh1 Configures PWM_CH1_INVERT register
- Returns:
  - None
- Post condition:
  - None

PWM_Ch2_Config(US cmpOn, US maxOn, US delayCh2, UC invertCh2)
PWM_Ch2_Config() – Configures PWM channel 2 registers.

- Parameters:
  - cmpOn Configures PWM_CH2_CMP_ON register
  - MaxOn Configures PWM_CH2_MAX_ON_TIME register
  - delayCh2 Configures PWM_CH2_DELAY register
  - invertCh2 Configures PWM_CH2_INVERT register
- Returns:
  - None
- Post condition:
  - None

PWM_Ch34_Config(US cmpOn, US maxOn, US delayCh34, UC invertCh34)
PWM_Ch34_Config() – Configures PWM channel 3/4 registers.

- Parameters:
  - cmpOn Configures PWM_CH34_CMP_ON register
  - MaxOn Configures PWM_CH34_MAX_ON_TIME register
  - delayCh34 Configures PWM_CH34_DELAY register
  - invertCh34 Configures PWM_CH34_INVERT register
- Returns:
  - None
- Post condition:
  - None

PWM_CH34_Output_Select(UC outputSelect)
PWM_CH34_Output_Select() – Select PWM34 output on PWM3 or PWM4pin.

- Parameters:
  - outputSelect Configures PIN_MUX for PWM34 output select
- Returns:
  - None
- Post condition:
  - None

9.17 WDT Functions

This section describes API functions of watch dog timer interface.

WDT_ENABLE()
WDT_ENABLE() – Enable watch dog timer.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

WDT_DISABLE()
WDT_DISABLE() – Disable watch dog timer.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

WDT_TRIGGER(x)
WDT_TIRGGER() – Load watch dog trigger value into WDOG_TRIG_HIGH register.

- Parameters:
  - x Data to be written to WDOG_TRIG_HIGH register
- Returns:
  - None
- Post condition:
  - None

WDT_TRIG_LOW_CONFIG(x)
WDT_TIRG_LOW_CONFIG() – Load watch dog trigger value into WDOG_TRIG_LOW register.

- Parameters:
  - x Data to be written to WDOG_TRIG_LOW register
- Returns:
  - None
- Post condition:
  - None

WDT_RESET_CLEAR()
WDT_RESET_CLEAR () – Clear watch dog reset bit by writing 1.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

WDT_TIMED_OUT()
WDT_TIMED_OUT () – Provides status of watchdog timed out.

- Parameters:
  - None
- Returns:
  - 1 if watchdog is timed out.
- Post condition:
  - None

9.18 Main Function

This section describes API function of main source file.

void main (void)
main() – Main routine.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

Main functions contains loop. Main loop calls pressure and temperature coefficient calculation function.

Pressure and temperature coefficients are calculated every 2.048 ms.

9.19 Filter Functions

This section describes API functions of FIR filter.

void FIR_Reset_Init (void)
FIR_Reset_Init() – Initializes all FIR filter related global variables.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

9.20 Application Functions

This section describes application specific functions.

void APP_Reset_Init(void)
APP_Reset_Init() – Initializes application specific global variables.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void APP_Calculate_Coeff (void)
APP_Calculate_Coeff() – State machine for T coefficient and P calculation.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

void CFG_Peripheral_Config (void)
CFG_Peripheral_Config() – Configuration of PGA970 peripherals.

- Parameters:
  - None
- Returns:
  - None
- Post condition:
  - None

Acronyms

Table A-1. List of Acronyms

TERM	DESCRIPTION
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
FW	Firmware
GPIO	General Purpose Input Output
HW, SW	Hardware, Software
ISR	Interrupt Service Routine
NA	Not Applicable
OEM	Original Equipment Manufacturer
OWI	One Wire Interface
SPI	Serial Peripheral Interface
TBD	To Be Decided

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265