Application Report
SLAA671A–August 2015–Revised May 2016

Generic Wrist Watch Programmer's Guide v0.2

ABSTRACT

This document provides information on the wristwatch RevD firmware architecture and the AFE44xx library. This document describes the basic architecture of the wristwatch firmware and how the AFE44xx library is integrated.

For more information on the AFE4403 Watch EVM, visit the tool folder at http://www.ti.com/tool/afe4403-hrevm.

Source code and related information can be downloaded from http://www.ti.com/lit/zip/slaa671.

Embedded Workbench is a trademark of IAR Systems.

Contents

List of Figures

List of Tables

1 Related Documentation

- *MSP430F551x, MSP430F552x Mixed-Signal Microcontrollers*
- *MSP430x5xx and MSP430x6xx Family User's Guide*
- Wrist Watch Data Logging Procedure v3.0
- TI Wrist Watch Demo v1.0

2 Required Software

- IAR Embedded Workbench™ for MSP430 v6.10.1 or newer
- Zip file with project collateral and source code

3 Software Installation

3.1 IAR Project

The zip file associated with this application report has three folders that contain source code (src), Doxygen documentation (doc), and the binary GNU make for windows (gmake). Two alternatives are provided to build this project, using a Makefile or using an IDE project. This application report only discusses the IDE project. For information on how to build the project using Makefiles, review the instructions in README.txt.

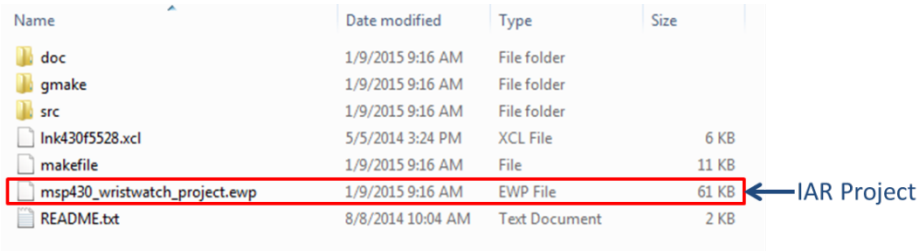


Figure 1. IAR Project Structure

The file for IAR Embedded Workbench project is located in root directory as shown in Figure 1.

To open the project in IAR Embedded Workbench:

1. Open the Project: Project → Add Existing Project… and select msp430_wrist watch_project.ewp.

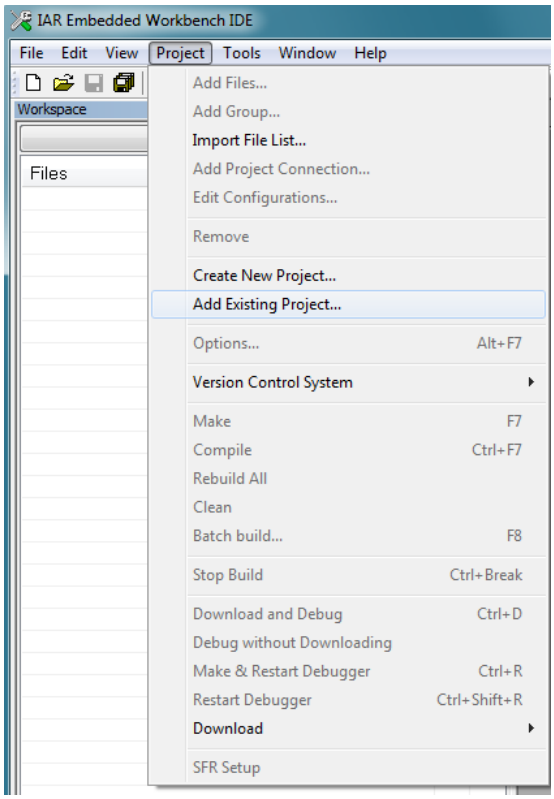


Figure 2. Open Project

2. Save the Workspace: File → Save Workspace and save it as msp430_wrist watch_project.eww.
3. Compile the project: Project → Make.
4. Download the code: Project → Download and Debug.
5. Run the code: Debug → Go.

The files provided in this project are resources to help get your project running on your target. You may use these files as is or use them as examples to create your own.

3.2 CCS Project

Similar to the IAR project, the associated zip file has three folders that contain CCS source code (src), doxygen documentation (doc), and the binary GNU make for windows (gmake).

The file for CCS project is located in root directory (see Figure 3).

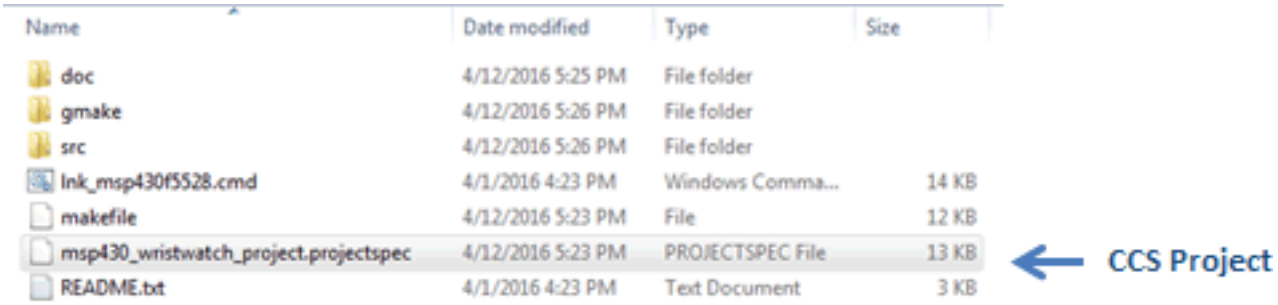


Figure 3. CCS Project Structure

To open the project in CCS:

1. Open the Project: Click Project → Import CCS Projects (see Figure 4) and browse to select the folder containing the CCS project file in Figure 3.

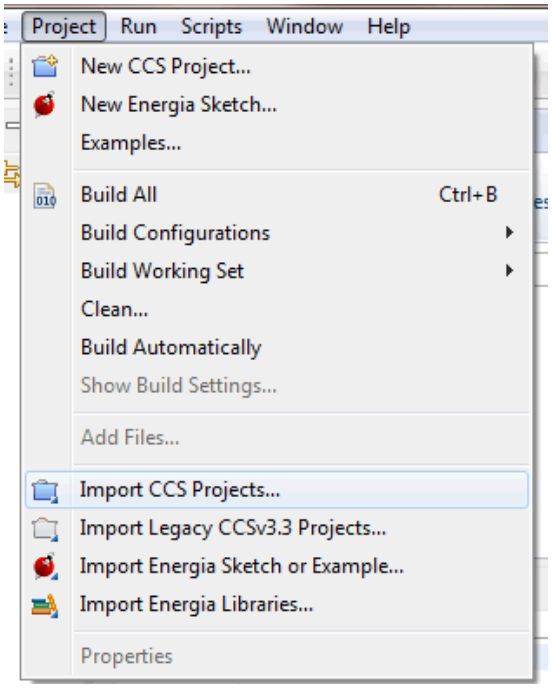


Figure 4. Import CCS Project

2. Click Finish (see Figure 5)

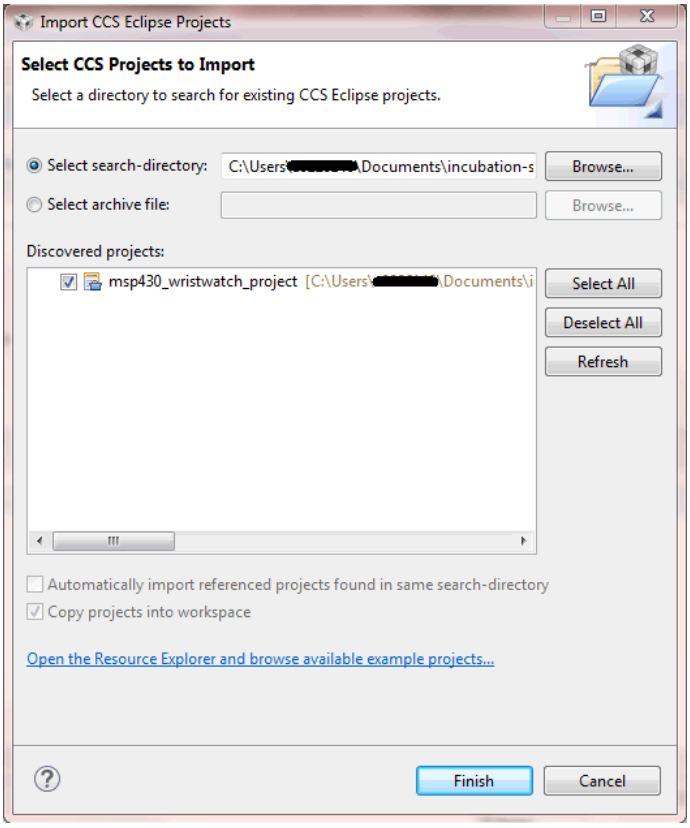


Figure 5. Select CCS Project to Import

3. Compile the project: Click Project → Build Project.
4. Download the code: Click Run → Debug.

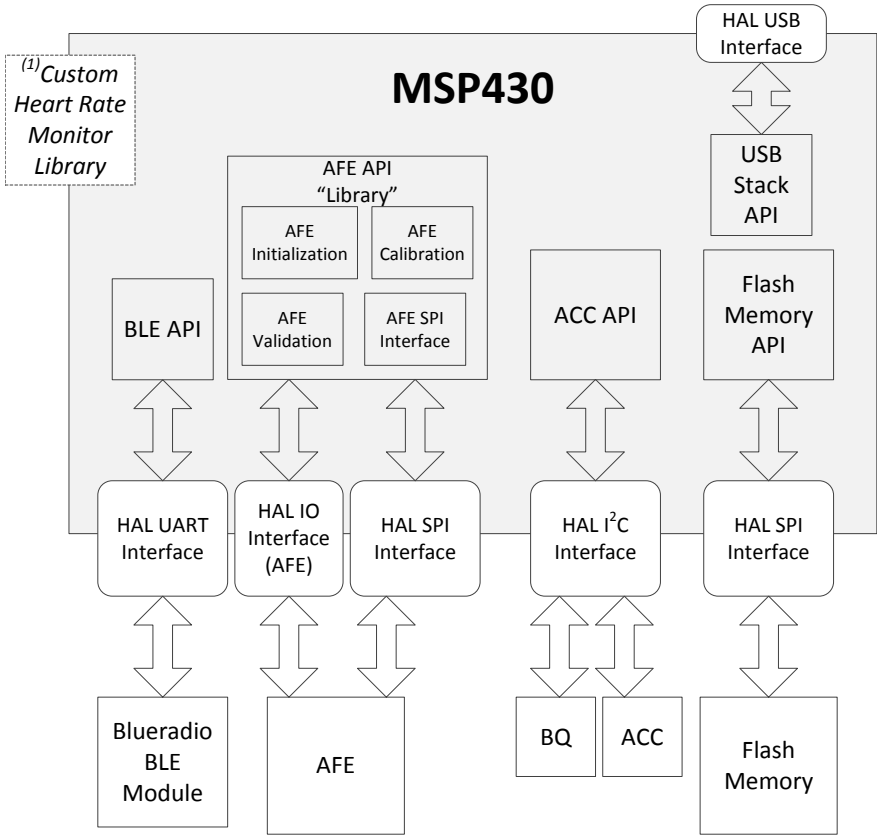
The files provided in this project are resources to help get your project running on your target. You may use these files as is or use them as examples to create your own.

4 Architecture

The wristwatch RevD uses a TI MSP430F5528 16-bit ultra-low-power microcontroller with 128KB of flash and 10KB of RAM.

4.1 System

The system block diagram in Figure 6 shows a high-level picture of the system and the software components inside the MSP430 device. The input and output data exchanged with the outside world are also shown.



(1): The Custom Heart Rate Monitor Library is not included in this package; therefore, it should be implemented and integrated by the user. The firmware provides hooks that can help with this integration.

Figure 6. System Block Diagram

The Hardware Abstraction Layer (HAL) takes any hardware-specific characteristics on the MSP430 MCU and provides a general set of interfaces to the physical components.

4.2 Firmware

The firmware is designed in a modular fashion to help with reuse of the firmware (see Figure 7). An example of this is that the HAL uses the MSP Driver Library (MSPDRIVERLIB) v1.95.xx.xx (refer to file "version.h" to get the actual driver library version) and the MSP USB STACK 4_20_00.

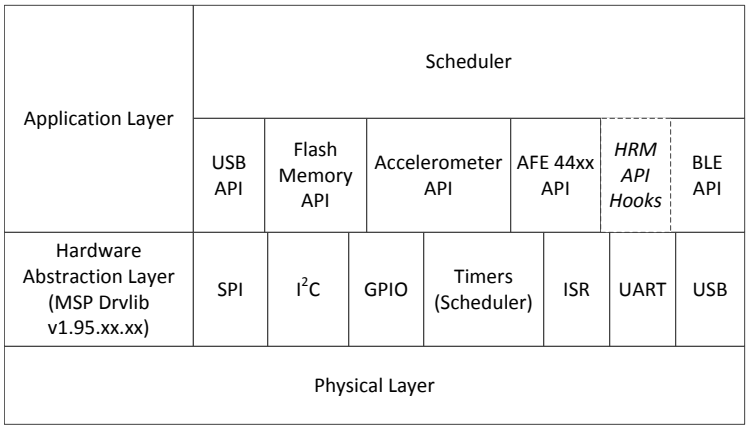


Figure 7. Software Design

5 Wrist Watch Application

5.1 Software Design

5.1.1 Wrist Watch State Machine

A state machine is implemented to control the functionality of the wrist watch (see Figure 8). The states of this state machine are:

- ***WRISTWATCH_INIT_STATE*** – Initialization state, erase external flash, configures GPIOs and UART, I^2C, and SPI peripherals
- ***WRISTWATCH_TOGGLE_LED_STATE*** – Toggles D2 (blue LED) and waits for the user to press the push button
- ***WRISTWATCH_PROCESS_PPG_STATE*** – Enables the AFE44xx and runs the Heart Rate Algorithm (if available). It also logs the data into external flash and it can store up to 6 hours of data.
- ***WRISTWATCH_IDLE_STATE*** – Disables the AFE, reconfigures heartbeat measurement, and switches to a different scheduler table (idle). At the same time, it enables the USB so the user can download the logged data.
- ***WRISTWATCH_BSL_STATE*** – Enables the bootstrap loader for firmware updates

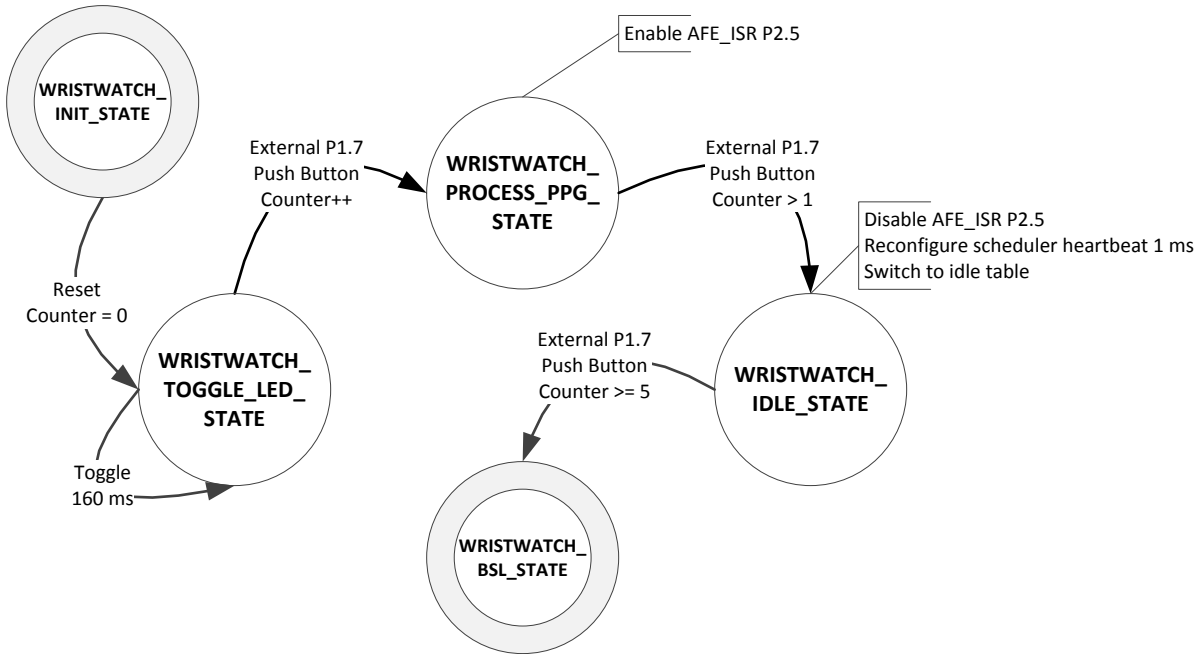


Figure 8. State Machine

The state machine is controlled by the user using the wrist watch push button (external interrupt) connected to P1.7. Figure 9 shows where the push button is located.

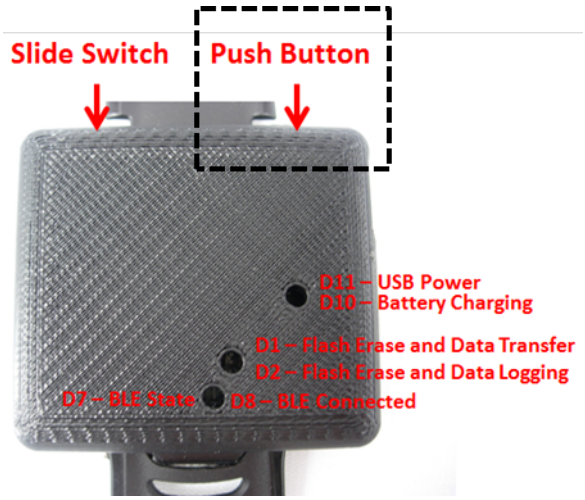


Figure 9. Wrist Watch Switches and LEDs

5.1.2 Main Flow Chart

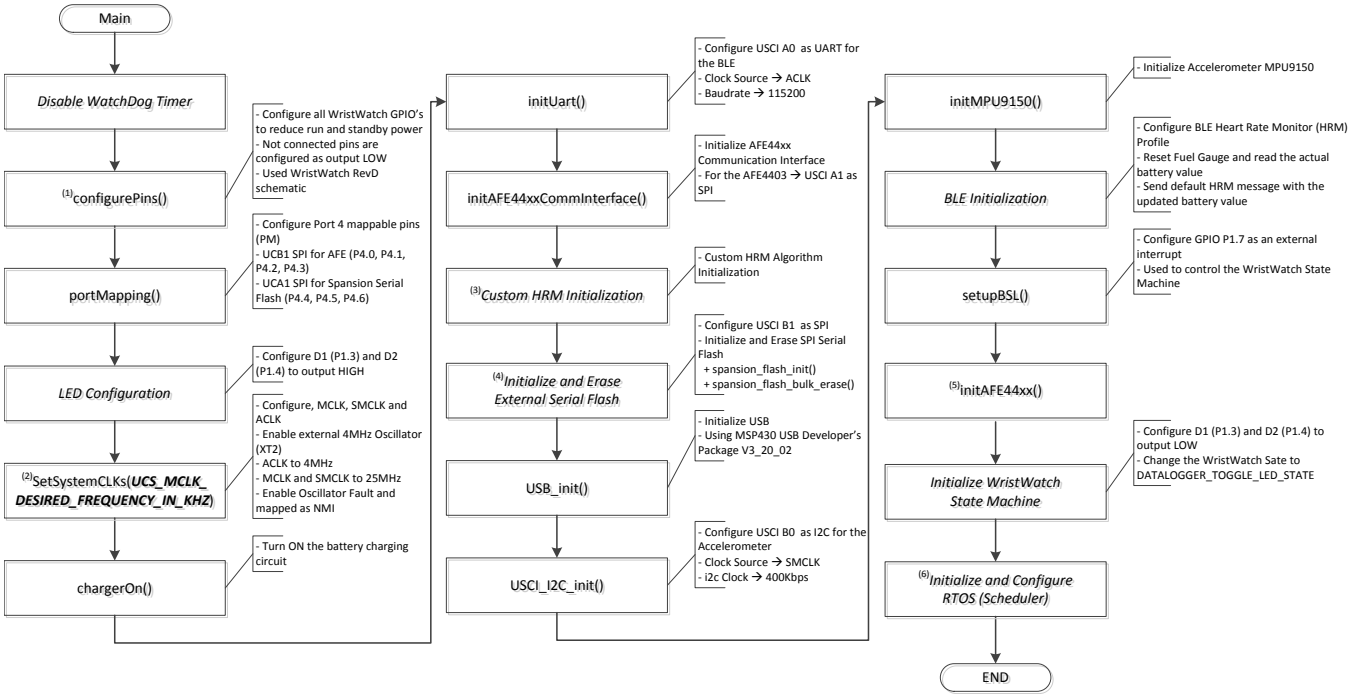


Figure 10. Main Routine

1. GPIO Configuration based on wrist watch RevD (see Table 1)
2. The ***UCS_MCLK_DESIRED_FREQUENCY_IN_KHZ*** is defined in SetMCLK.h. By default it set to 25 MHz, but it can be changed to 16 MHz. See Section 5.3 to understand the difference between these configurations.
3. Place holder to call custom HRM algorithm initialization.
4. Initializing and erasing the external flash takes approximately 50 seconds to complete. If the erase fails, an LED sequence is displayed (toggle between LED D1 and LED D2), and the logging is disabled until the next power on cycle. The heart rate is still reported through the BLE message.
5. This routine is explained in Section 5.2.1.
6. The scheduler is explained in Section 5.1.3.

Table 1. MSP430 GPIO Configuration

Port	Signal	Configuration		Port	Signal	Configuration	
P1.0	ACLK_MSP430	OUTPUT	LOW	P4.0	SPI_AFE_STE	OUTPUT	LOW
P1.1	LDO_OUT_FG/ BAT_DET	INPUT	HIGH	P4.1	SPI_AFE_SIMO	OUTPUT	LOW
P1.2	GPOUT	INPUT	HIGH	P4.2	SPI_AFE_SOMI	OUTPUT	LOW
P1.3	LED1	OUTPUT	LOW	P4.3	SPI_AFE_CLK	OUTPUT	LOW
P1.4	LED2	OUTPUT	LOW	P4.4	SPI_MEM_SI	OUTPUT	LOW
P1.5	UART_CTS_BLE	OUTPUT	LOW	P4.5	SPI_MEM_SO	INPUT	HIGH
P1.6	UART_RTS_BLE	OUTPUT	LOW	P4.6	SPI_MEM_CLK	OUTPUT	LOW
P1.7	INT_PB1	INPUT	LOW	P4.7	NC	OUTPUT	LOW
P2.0	NC	OUTPUT	LOW	P5.0	NC	OUTPUT	LOW
P2.1	INT_MPU	OUTPUT	LOW	P5.1	NC	OUTPUT	LOW
P2.2	AFE_DIAG_END	OUTPUT	LOW	P5.2	XT2IN	OUTPUT	LOW
P2.3	AFE_RESETZ	INPUT	HIGH	P5.3	XT2OUT		
P2.4	AFE_PDNZ	INPUT	HIGH	P5.4	NC	OUTPUT	LOW
P2.5	AFE_ADC_RDY	OUTPUT	LOW	P5.5	NC	OUTPUT	LOW
P2.6	SPI_MEM_WP	INPUT	HIGH				
P2.7	SPI_MEM_HOLD	OUTPUT	LOW	P6.0	CHRGR_CE	INPUT	HIGH
				P6.1	CHRGR_EN1_EN2	OUTPUT	LOW
P3.0	I2C_SDA	INPUT	HIGH	P6.2	NC	OUTPUT	LOW
P3.1	I2C_SCL	INPUT	HIGH	P6.3	NC	OUTPUT	LOW
P3.2	NC	OUTPUT	LOW	P6.4	SPI_MEM0_CS	INPUT	HIGH
P3.3	UART_RX_BLE	INPUT	HIGH	P6.5	SPI_MEM1_CS	INPUT	HIGH
P3.4	UART_TX_BLE	OUTPUT	LOW	P6.6	NC	OUTPUT	LOW
				P6.7	NC	OUTPUT	LOW
PJ.0	JTAG_TDO	OUTPUT	LOW				
PJ.1	JTAG_TDI	OUTPUT	LOW				
PJ.2	JTAG_TMS	OUTPUT	LOW				
PJ.3	JTAG_TCK	OUTPUT	LOW				

5.1.3 Scheduler

The scheduler that is used in this project can support up to eight different tasks per table and a background task. Each task rate is a power of 2 multiple (with a max of 2^8) of the heartbeat and the tables are configured in os_tasks.c. Table 2 lists examples rates using 5-ms and 1-ms heartbeats.

Table 2. Scheduler Example Rates

2^0	1	×	5	=	5	ms	2^0	1	×	1	=	1	ms
2^1	2	×	5	=	10	ms	2^1	2	×	1	=	2	ms
2^2	4	×	5	=	20	ms	2^2	4	×	1	=	4	ms
2^3	8	×	5	=	40	ms	2^3	8	×	1	=	8	ms
2^4	16	×	5	=	80	ms	2^4	16	×	1	=	16	ms
2^5	32	×	5	=	160	ms	2^5	32	×	1	=	32	ms
2^6	64	×	5	=	320	ms	2^6	64	×	1	=	64	ms
2^7	128	×	5	=	640	ms	2^7	128	×	1	=	128	ms
2^8	256	×	5	=	1280	ms	2^8	256	×	1	=	256	ms

Depending on the wrist watch state, the heartbeat is configured to either 1 ms or 5 ms. Figure 11 shows the scheduler table that will be running in each of the states.

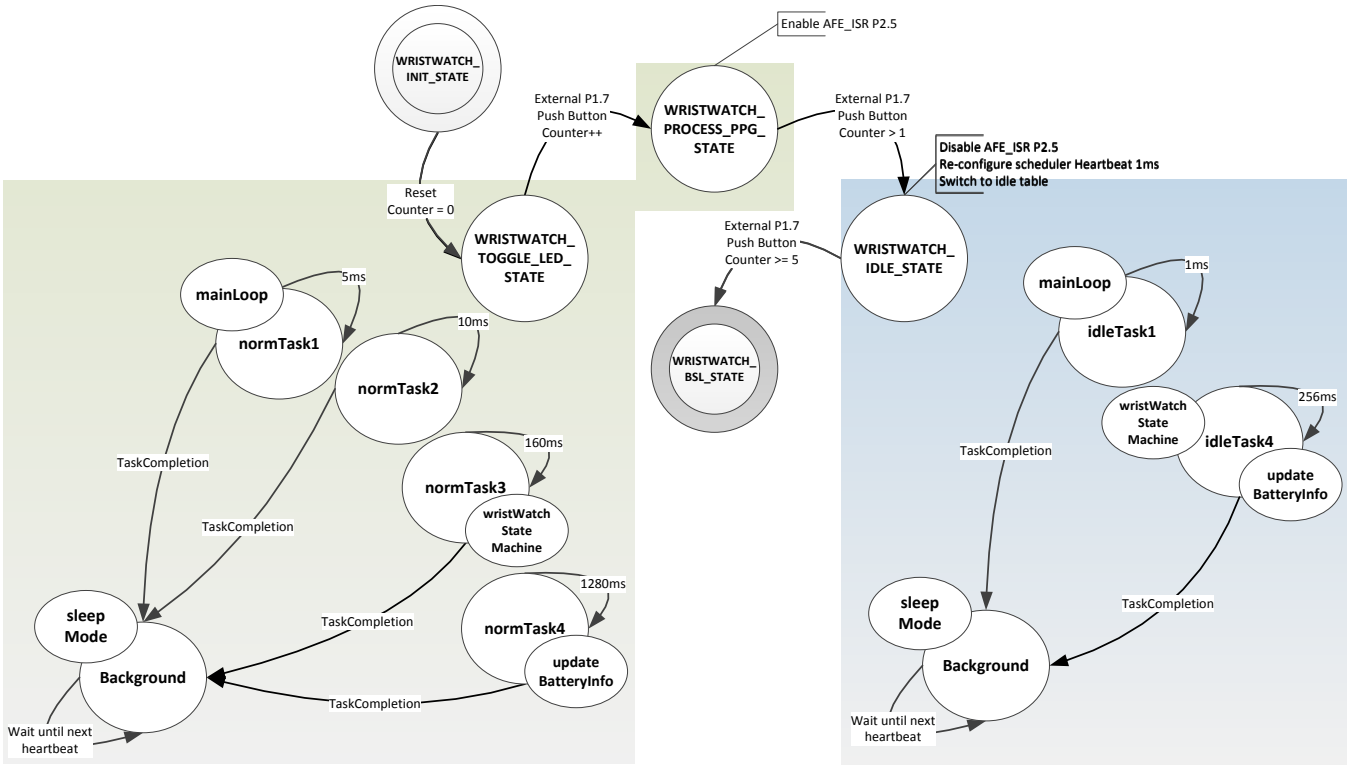


Figure 11. State Machine and Scheduler Tables

For this application, two tables are defined, one to be used in normal mode (see Section 5.1.3.1) and the other one to be used in idle mode (see Section 5.1.3.2). Neither table includes all eight tasks.

5.1.3.1 Normal Mode Table

This table only has four tasks.

```
OS_taskType osTaskListNorm[NUM_TASKS_IN_TABLE] =
{       /* rate  offset  action  */
        {    0,     0,   &normTask1 },    /* highest priority task */ /*    5.0 ms */
        {    1,     1,   &normTask2 },                                 /*   10.0 ms */
        {   31,     7,   &normTask3 },                                 /*  160.0 ms */
        {  255,   127,   &normTask4 },    /* lowest priority task */  /* 1280.0 ms */
};
```

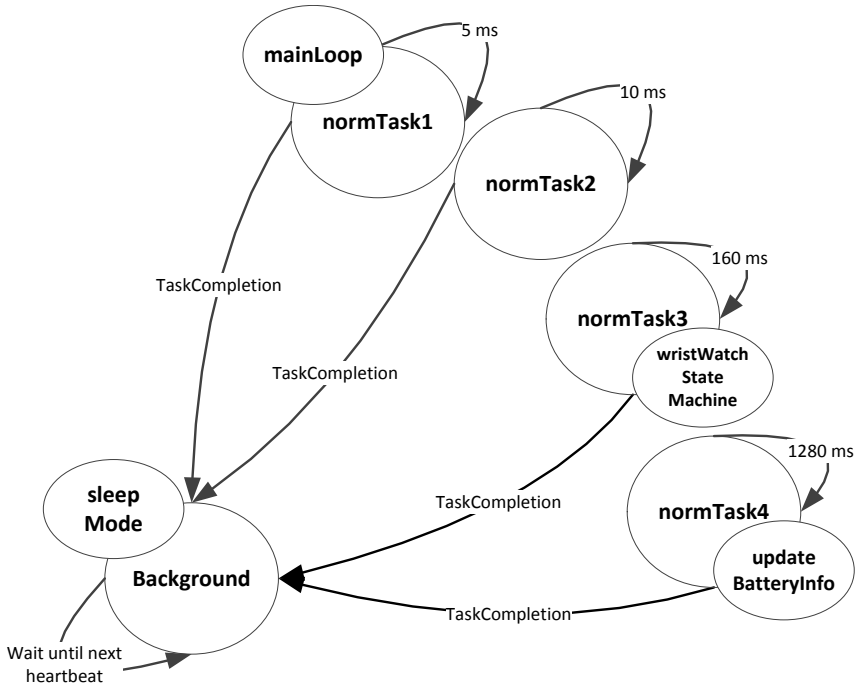


Figure 12. Scheduler With Normal Mode Tasks

5.1.3.1.1 normTask1

This task runs every 5 ms and calls the mainLoop function only when the wristWatchState is different from WRISTWATCH_TOGGLE_LED_STATE.

- mainLoop
  - If needed, there is a hook for a Spot HRM function. Currently it is configured to be called for only the first 15 second (using _15_SECONDS_5ms constant in "os_tasks.h") after the wristWatchState is changed to WRISTWATCH_PROCESS_PPG_STATE. This could reconfigured by the user.
  - The 15-second timer is also part of Task1. 5 ms is the time base for the constant calculation.
  - Runs the USB state machine, but the logging capabilities are not enabled until wristWatchState is changed to WRISTWATCH_IDLE_STATE.

5.1.3.1.2 normTask2

This task runs every 10 ms. Currently it is not used, but it is available for future use.

5.1.3.1.3 normTask3

This task runs every 160 ms and calls the wristWatchStateMachine function.

- wristWatchStateMachine
  - The state machine is updated every time that wrist watch push button is pressed.

5.1.3.1.4 normTask4

This task runs every 1280 ms and, depending on the wrist watch state, updates the BLE battery service message with the battery information or with CPU throughput (if MEASURED_THROUGHPUT is enabled; refer to Section 5.1.3.3).

- updateBatteryInfo
  - Updates battery value in the BLE message

5.1.3.2 Idle Mode Table

This table has only two tasks and is selected when wristWatchState is changed to WRISTWATCH_IDLE_STATE.

```
OS_taskType osTaskListIdle[NUM_TASKS_IN_TABLE_IDLE] =
{        /* rate  offset  action  */
        {    0,     0,   &idleTask1 },    /* highest priority task */ /*    1.0 ms */
        {  255,   127,   &idleTask4 },    /* lowest priority task */  /*  256.0 ms */
};
```

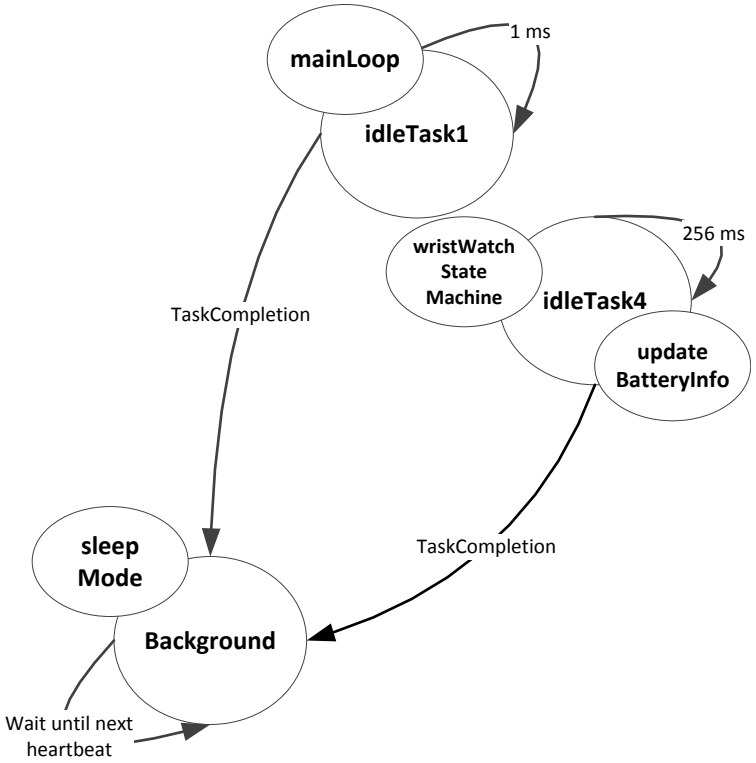


Figure 13. Scheduler With Idle Mode Tasks

5.1.3.2.1 idleTask1

This task runs every 1 ms and calls the mainLoop.

- mainLoop
  - Runs the USB state machine, enabling the user to download the logged data.

5.1.3.2.2 idleTask4

This task runs every 256 ms and updates the BLE battery service message with the battery information or with CPU throughput, if the MEASURED_THROUGHPUT is enabled (see Section 5.1.3.3).

- wristWatchStateMachine
  - The state machine is updated every time that wrist watch push button is pressed.
- updateBatteryInfo
  - Updates battery value in the BLE message

5.1.3.3 Background Task

This subroutine is called when no other task is scheduled. It can measure the CPU throughput of the system. In other words, the CPU remains here in sleep mode (low-power mode 0) until the next heartbeat. By default, the CPU throughput measurement is not enabled. To enable it, uncomment the MEASURED_THROUGHPUT define in prg_info.h. After enabling this define, rebuild the application.

The way that the CPU throughput is calculated is very simple: a timer is used to calculate the CPU time spent in sleep mode (LPM0) and, after a period of time (500 ms), an average is calculated. This is the average sent through the BLE message in normTask4 or idleTask4.

The battery level is not displayed when CPU throughput calculation is enabled. Figure 14 and Figure 15 show the CPU throughput measurement using the TI BLE Multitool app. Refer to the *Wrist Watch Data Logging Procedure* for more information about this tool.

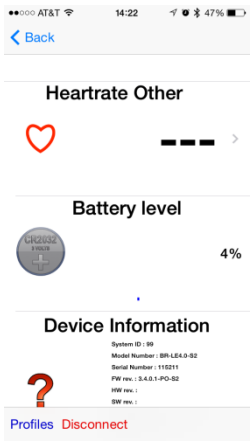


Figure 14. WRISTWATCH_TOGGLE_LED_STATE

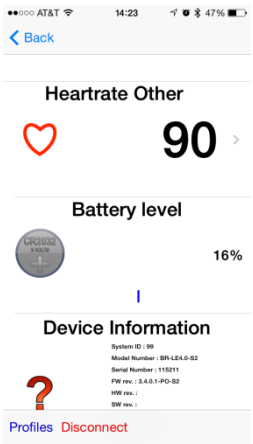


Figure 15. WRISTWATCH_PROCESS_PPG_STATE

5.2 Libraries

5.2.1 AFE44xx

The AFE44xx binary library for this application report is called msp430_afe4403_lib.r43. This library can be configured using the files config_AFE44xx.c and config_AFE44xx.h.

The AFE44xx API calls are located in AFE44xx.h and in config_AFE44xx.h. The file config_AFE44xx.c contains the GPIO and communication interface configurations between the MSP430 MCU and the AFE44xx.

5.2.1.1 AFE44xx GPIO and Communication Configuration (config_AFE44xx.c)

All of the GPIOs that are used between the MSP430 MCU and the AFE44xx are configured using the following data types:

- AFE44xx Pins
  - RESETZ, AFE_PDNZ, DIAG_END
  - LEDs D1 (STREAM_STATUS_LED) and D2 (PACKET_TX_STATUS_LED)

```
struct AFE44xx_GPIO{
  uint8_t selectedPort;
  uint16_t selectedPin; };
```

- AFE44xx Pins as Interrupt
  - ADC_RDY

```
struct AFE44xx_PORT_ISR{
  uint8_t selectedPort;
  uint16_t selectedPin; };
```

- AFE44xx SPI Interface
  - SPISTE, SCLK, SPISOMI, SPISIMO

```
struct AFE44xx_SPI{
  uint16_t baseAddress;
  uint8_t  selectClockSource;
  uint32_t desiredSpiClock;
  uint8_t  msbFirst;
  uint8_t  clockPhase;
  uint8_t  clockPolarity;
  struct   AFE44xx_GPIO SPI_Tx;
  struct   AFE44xx_GPIO SPI_Rx;
  struct   AFE44xx_GPIO SPI_Clk;
  struct   AFE44xx_GPIO SPI_CS; };
```

For the SPI interface, some other parameters need to be specified (see Table 3)

Table 3. SPI Parameters

Parameter	Description
baseAddress	The base address of the SPI module.
selectClockSource	The frequency of the selected clock source
desiredSpiClock	The desired clock rate for SPI communication
clockPhase	Clock phase select. Valid values are: USCI_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT (Default) USCI_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT
clockPolarity	Clock polarity select. Valid values are: USCI_SPI_CLOCKPOLARITY_INACTIVITY_HIGH USCI_SPI_CLOCKPOLARITY_INACTIVITY_LOW (Default)

5.2.1.2 AFE44xx APIs (config_AFE44xx.h, AFE44xx.h)

The following APIs are provided:

config_AFE44xx.h

- void readAccelerometerData(uint8_t *data)
  - Callback function to read the accelerometer
  - Called inside the library by afe44xxDRDYIsr

  Parameters

  data – pointer to the accelerometer buffer

  Returns

  None
- void initAFE44xxCommInterface (void)
  - Configures the AFE44xx communication interface
  - By default uses the SPI parameters specified in the structure "AFE44xx_SPI"
  - Called in main

  Parameters

  None

  Returns

  None

AFE44xx.h

- void initAFE44xx(void)
  - Initializes and calibrate the AFE44xx device
  - Called in main

    Parameters

    None

    Returns

    None
- void afe44xxDRDYIsr (void)
  - Interrupt handler called when the AFE44xx DRDY signal is received
  - Reads AFE44xx registers, accelerometer data and evaluate the data
  - Allows processPPG (HRM Custom Algorithm) to run
  - It also stores all this information including the calculated heart rate in the external serial flash using the dataLogger function, and it also reads the battery information and sends the BLE Heart Rate Message
  - Called in Port2_ISR

    Parameters

    None

    Returns

    None
- void processPPG(void)
  - Callback/Hook function to run the HRM algorithm
  - Called in processPPGTask
  - Make sure to store the updated Heart Rate in the variable g_ui16HeartRate

    Parameters

    None

    Returns

    None
- uint16_t g_ui16HeartRate
  - Variable that stores the value of the Heart Rate
  - Defined inside of the library because of logging purposes
  - For testing purposes (***WRISTWATCH_PROCESS_PPG_STATE)***, this variable is incremented in normTask4 (os_tasks.c). Remove this code when integrating your HRM algorithm.

```
if (g_ui16HeartRate > 180)
{
    g_ui16HeartRate = 0;
}
g_ui16HeartRate++;
```

5.2.1.3 Outside Functions Called by the Library

- void enablePPGTimer(void)
  - Configures and enables Timer B0 to interrupt every 1 ms
  - Uses the SMCLK as clock source
  - Called in afe44xxDRDYIsr

  Parameters

  None

  Returns

  None

- uint8_t dataLogger(void)
  - Stores information into the external serial flash
  - Called in afe44xxDRDYIsr

  Parameters

  None

  Returns

  0 = Continue Logging

  1 = Stop Logging "Memory Full"

This library starts running when the state WRISTWATCH_PROCESS_PPG_STATE is selected. Figure 16 shows an overview of the library data flow.

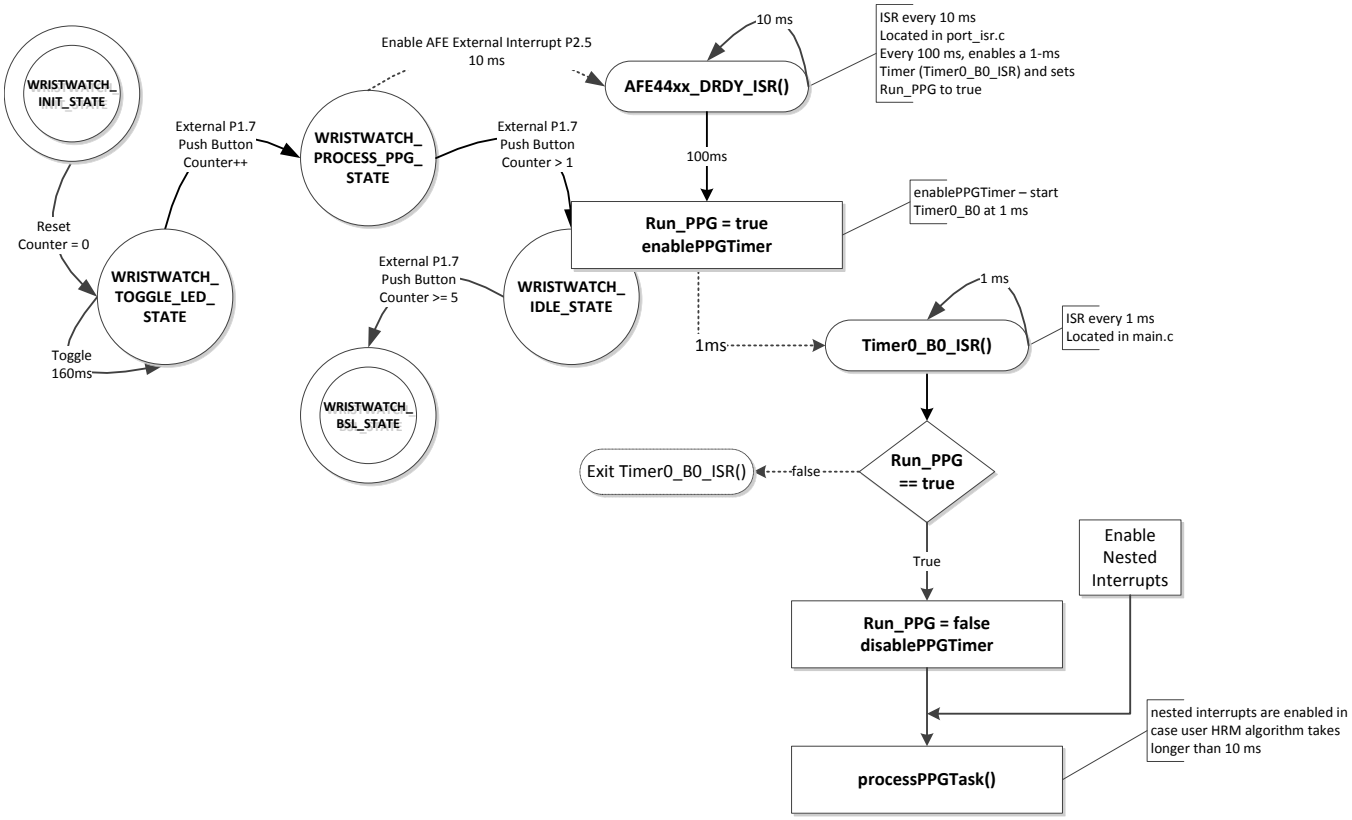


Figure 16. Cross Functional AFE44xx Flow Chart

5.3 Measurements

The following measurements were taken on each of the wrist watch states using LPM0 over the recommended ranges of supply voltage and operating free-air temperatures.

5.3.1 IAR Measurements

Table 4 and Table 5 summarize the current consumption of the IAR project running at 25 MHz and 16 MHz, respectively.

Table 4. Running at 25 MHz (PMMCOREV3) (Default Configuration)

WRISTWATCH_ INIT_STATE[1]		WRISTWATCH_ TOGGLE_LED_STATE		WRISTWATCH_ PROCESS_PPG_STATE[2]		WRISTWATCH_IDLE_STATE	
CPU Throughput		CPU Throughput		CPU Throughput		CPU Throughput	
Run	Sleep	Run	Sleep	Run	Sleep	Run	Sleep
100%	0%	4%	96%	5%	95%	5%	95%
9.26 mA		***1.04 mA***		***1.12 mA***		***0.90 mA***	

(1) WRISTWATCH_INIT_STATE requires approximately 50 seconds to complete (waits until it erases the external serial flash).
(2) HRMWRISTWATCH_PROCESS_PPG_STATE depends on the HRM algorithm.

Table 5. Running at 16 MHz (PMMCOREV2)

WRISTWATCH_ INIT_STATE[1]		WRISTWATCH_ TOGGLE_LED_STATE		WRISTWATCH_ PROCESS_PPG_STATE[2]		WRISTWATCH_IDLE_STATE	
CPU Throughput		CPU Throughput		CPU Throughput		CPU Throughput	
Run	Sleep	Run	Sleep	Run	Sleep	Run	Sleep
100%	0%	5%	95%	6%	94%	6%	94%
6.20 mA		***1.00 mA***		***1.10 mA***		***0.87 mA***	

(1) WRISTWATCH_INIT_STATE requires approximately 50 seconds to complete (waits until it erases the external serial flash).
(2) HRMWRISTWATCH_PROCESS_PPG_STATE depends on the HRM algorithm.

5.3.2 CCS Measurements

Table 6 and Table 7 summarize the current consumption of the CCS project running at 25 MHz and 16 MHz, respectively.

Table 6. Running at 25 MHz (PMMCOREV3) (Default Configuration)

WRISTWATCH_ INIT_STATE[1]		WRISTWATCH_ TOGGLE_LED_STATE		WRISTWATCH_ PROCESS_PPG_STATE[2]		WRISTWATCH_IDLE_STATE	
CPU Throughput		CPU Throughput		CPU Throughput		CPU Throughput	
Run	Sleep	Run	Sleep	Run	Sleep	Run	Sleep
100%	0%	4%	96%	5%	95%	5%	95%
9.30 mA		***1.23 mA***		***2.11 mA***		***1.03 mA***	

(1) WRISTWATCH_INIT_STATE requires approximately 50 seconds to complete (waits until it erases the external serial flash).
(2) HRMWRISTWATCH_PROCESS_PPG_STATE depends on the HRM algorithm.

Table 7. Running at 16 MHz (PMMCOREV2)

WRISTWATCH_ INIT_STATE[1]		WRISTWATCH_ TOGGLE_LED_STATE		WRISTWATCH_ PROCESS_PPG_STATE[2]		WRISTWATCH_IDLE_STATE	
CPU Throughput		CPU Throughput		CPU Throughput		CPU Throughput	
Run	Sleep	Run	Sleep	Run	Sleep	Run	Sleep
100%	0%	5%	95%	6%	94%	6%	94%
6.04 mA		***1.11 mA***		***1.84 mA***		***0.93 mA***	

(1) WRISTWATCH_INIT_STATE requires approximately 50 seconds to complete (waits until it erases the external serial flash).
(2) HRMWRISTWATCH_PROCESS_PPG_STATE depends on the HRM algorithm.

5.3.3 Memory Footprint for IAR Project

The memory sizes listed in Table 8 and Table 9 are based on the optimizations in Figure 17.

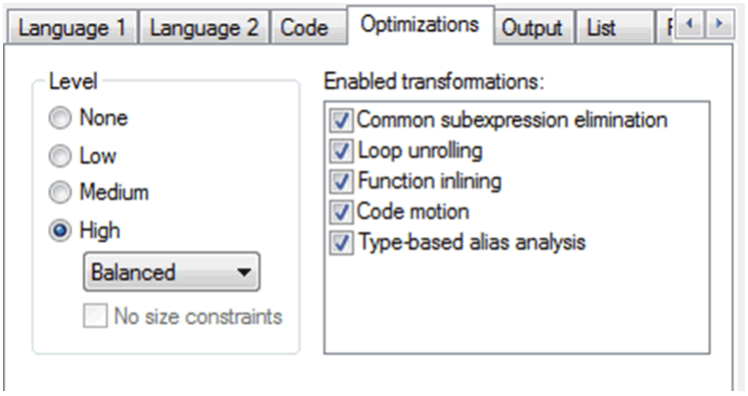


Figure 17. IAR Optimizations

Table 8 summarizes the memory footprint (includes the msp430_afe4403_lib).

Table 8. Application Memory Footprint

Flash		RAM
Code	**Constants**	**Data**
19660 bytes	1153 bytes	3826 bytes

Table 9 summarizes the memory footprint for the msp430_afe4403_lib.

Table 9. Library Memory Footprint

Flash		RAM
Code	**Constants**	**Data**
2366 bytes	19 bytes	70 bytes

5.3.4 Memory Footprint for CCS Project

The memory sizes listed in Table 10 and Table 11 are based on the optimizations in Figure 18.

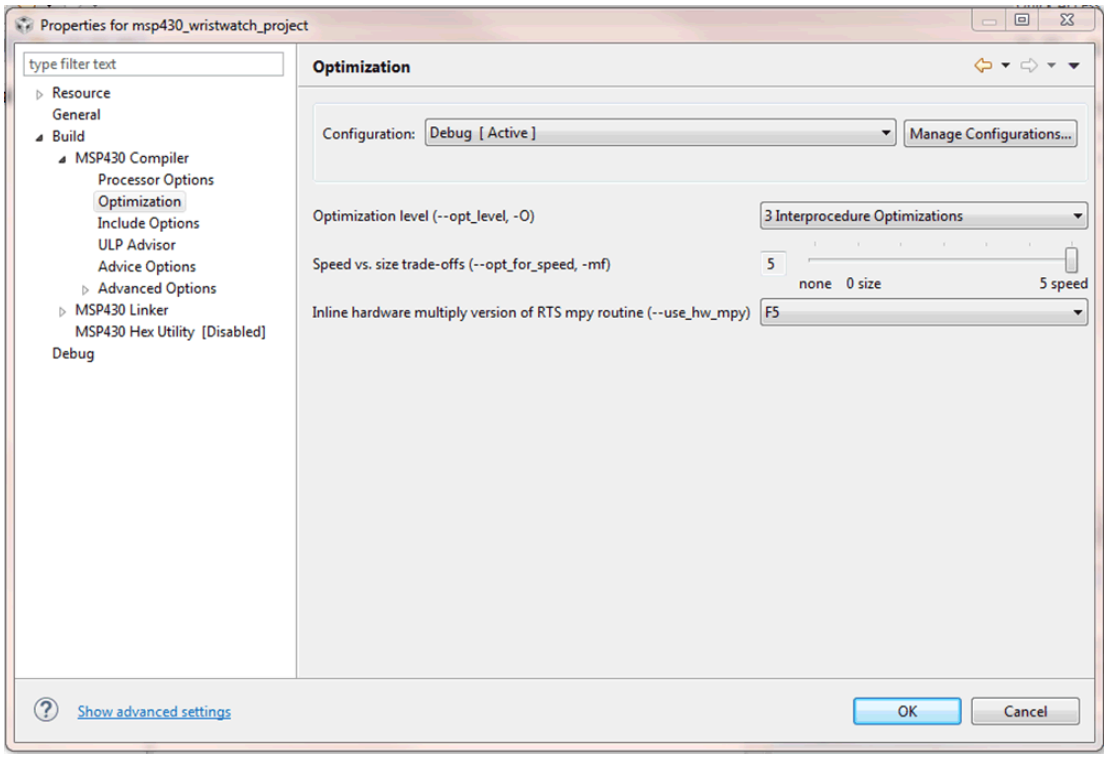


Figure 18. CCS Optimizations

Table 10 summarizes the memory footprint (includes the msp430_afe4403_lib).

Table 10. Application Memory Footprint

Flash		RAM
Code	Constants	Data
28790 bytes	944 bytes	4911 bytes

Table 11 summarizes the memory footprint for the msp430_afe4403_lib.

Table 11. Library Memory Footprint

Flash		RAM
Code	Constants	Data
3960 bytes	15 bytes	202 bytes

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from August 5, 2015 to May 31, 2016 **Page**

- Added Section 3.2, *CCS Project* 5
- Added Section 5.3.2, *CCS Measurements* 19
- Changed the code size in Table 8, *Application Memory Footprint* 20
- Changed the code size in Table 9, *Library Memory Footprint* 20
- Added Section 5.3.4, *Memory Footprint for CCS Project* 21

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