Application Report
SLAA538 – May 2012

Energy Meter Code Library for 1-Phase to 3-Phase Using MSP430 Family

Mekre Mesganaw *Metering Applications*

ABSTRACT

This application report describes how to execute the Texas Instruments MSP430 Energy Library, which uses a common set of source files to support meters based on the MSP430FE427A, MSP430F47197, MSP430F4794, MSP430F6736, and MSP430AFE253 devices. This application report includes the necessary information about the APIs of the energy library.

The MSP430 Energy Library is available here: http://www.ti.com/tool/msp430-energy-library.

WARNING

Failure to adhere to these steps and/or not heed the safety requirements at each step may lead to shock, injury, and damage to the hardware. Texas Instruments is not responsible or liable in any way for shock, injury, or damage caused due to negligence or failure to heed advice.

Contents

List of Figures

1 Introduction

The MSP430 Energy Library is the latest metering software package, which has support for the MSP430FE427A, MSP430F47197, MSP430F4794, MSP430F6736, and MSP430AFE253 metering devices. For each EVM, the energy metrology software is comprised of three projects. The first project is the toolkit library which contains mostly mathematics routines. The second project is the metrology library which calculates the metering parameters. The metrology library consists of a background process that collects voltage and current samples, calculates working parameters needed to calculate the final metering parameters (for example, RMS voltage, current, and frequency), and outputs energy-proportional pulses. When approximately one second worth of samples have been obtained, the background process asserts a flag to indicate that a new set of metering parameters are ready to be calculated. The third project of the metrology software is the application project, which is the code that actually runs on the EVM. When the background process of the metering library asserts the flag to indicate that a new set of metering parameters are ready to be calculated, the application project calls the function that calculates the metering parameters using the working parameters calculated by the metering library's background process. The application project also deals with UART communication, LCD (if available) support, multi-tariff support, and RTC support.

With the exclusion of three files, the same source files are shared among all the meters. Two of the excluded three files, metrology-parms.h and emeter-template.h, are used to configure meter features (for example, VRMS_SUPPORT, TEMPERATURE_SUPPORT, RTC_SUPPORT). The third excluded file is used to configure the LCD.

2 Function Description

2.1 Toolkit Project

void accum48(register int16_t x[3], register int32_t y)	
Parameters	*x* - 48-bit number where accumulation takes place. It is represented as a 3-element 16-bit array. *y* - 32-bit number to be added to x.
Returns	–
Description	Replaces a 48-bit number (x) with the sum of its current value and a 32-bit number (y); that is, x = y + x.
File	accum48.s43
Comments	–

void bin2bcd16(register uint8_t bcd[3], register uint16_t bin)	
Parameters	*bcd* - Result array that stores the BCD representation of the binary number. Each element in the array stores the BCD representation of two digits of the binary number. *bin* - 16-bit binary number to be converted to bcd format.
Returns	--
Description	Converts a 16-bit binary number into a binary coded decimal. The most significant digit is stored in the lower nibble of bcd[0] and the least significant digit is stored in the lower nibble of bcd[2].
File	bin2bcd16.s43
Comments	The upper nibble of bcd[0] is not needed because it would represent the sixth digit of a 16-bit number. Because only 5 digits are needed to represent the maximum value of a 16-bit value (65535), it is unnecessary.

void bin2bcd32(uint8_t bcd[5], uint32_t bin)	
Parameters	*bcd* - Result array that stores the BCD representation of the binary number. Each element in the array stores the BCD representation of two digits of the binary number. *bin* - 32-bit binary number to be converted to bcd format.
Returns	--
Description	Converts a 32-bit binary number into a binary coded decimal. The most significant digit is stored in the upper nibble of bcd[0] and the least significant digit is stored in the lower nibble of bcd[4].
File	bin2bcd32.s43
Comments	--

void bin2bcd64(uint8_t bcd[10], uint64_t bin)	
Parameters	*bcd* - Result array that stores the BCD representation of the binary number. Each element in the array stores the BCD representation of two digits of the binary number. *bin* - 64-bit binary number to be converted to bcd format.
Returns	--
Description	Converts a 64-bit binary number into a binary coded decimal. The most significant digit is stored in the upper nibble of bcd[0] and the least significant digit is stored in the lower nibble of bcd[9].
File	bin2bcd64.s43
Comments	--

*int16_t dc_filter16(int32_t *p, int16_t x)*	
Parameters	*p* - Pointer to 32-bit DC estimate of the waveform signal. *x* - 16-bit sample-reading of AC mains waveform signal before the DC component is removed.
Returns	16-bit sample reading of AC mains waveform signal with the DC component removed.
Description	Filters away the DC content from an AC mains waveform signal by using a heavily damped integrator to estimate the DC level. The current DC level is then subtracted from the signal.
File	dc_filter16.s43
Comments	This is not a generic DC filter. This function should be used on a channel that is running in 16-bit mode.

*void dc_filter16_init(int32_t *p, int16_t x)*	
Parameters	*p* - Pointer to DC estimate. *x* - Initial DC estimate used to prime a Mains signal's DC estimate. This value is set during the calibration process, based on the DC estimate measured at that time.
Returns	--
Description	Initializes a Mains signal's DC estimate, to ensure quick settling when the meter is powered up.
File	dc_filter16.s43
Comments	--

int32_t dc_filter24(int16_t p[3], int32_t x)	
Parameters	*p* - Pointer to DC estimate of the waveform signal. *x* - 24-bit sample-reading of AC mains waveform signal before the DC component is removed.
Returns	24-bit sample reading of AC mains waveform signal with the DC component removed. The 24-bit value is stored in a 32-bit int.
Description	Filter away the DC content from an AC mains waveform signal by using a heavily damped integrator to estimate the DC level. The current DC level is then subtracted from the signal.
File	dc_filter24.s43
Comments	This is not a generic DC filter. This function should be used on a channel that is running in 24-bit mode.

*void dc_filter24_init(int32_t *p, int16_t x)*	
Parameters	*p* - Pointer to DC estimate. *x* - Initial DC estimate used to prime a mains signal's DC estimate. This value is set during the calibration process, based on the DC estimate measured at that time.
Returns	--
Description	Initializes a Mains signal's DC estimate, to ensure quick settling when the meter is powered up.
File	dc_filter24.s43
Comments	--

int16_t dds_lookup(uint32_t phase)	
Parameters	*phase* - The 32-bit number corresponding to the phase to be looked up where the 32-bit integer-range maps to the 0-360° range. As an example, a value of 0x40000000 corresponds to a phase of 90°.
Returns	The amplitude of the sine wave at the specified phase.
Description	Look up the amplitude of a sine wave at a specified phase.
File	dds.c
Comments	--

int16_t dds_interpolated_lookup(uint32_t phase)	
Parameters	*phase* - The 32-bit number corresponding to the phase to be looked up where the 32-bit integer-range maps to the 0-360° range. As an example, a value of 0x40000000 corresponds to a phase of 90°.
Returns	The amplitude of the sine wave at the specified phase.
Description	Look up the amplitude of a sine wave at a specified phase using interpolation.
File	dds.c
Comments	--

*int16_t dds(uint32_t *phase_acc, int32_t phase_rate)*	
Parameters	*phase* - The 32-bit number corresponding to the phase to be looked up where the 32-bit integer-range maps to the 0-360° range. As an example, a value of 0x40000000 corresponds to a phase of 90°. *phase_rate* - The per sample phase increment.
Returns	The amplitude of the sine wave, as a 16-bit signed number.
Description	Performs direct digital sine wave synthesis.
File	dds.c
Comments	--

int32_t div48(int16_t x[3], int16_t y)	
Parameters	*x* - The 48-bit number to be divided. *y* - The 16-bit integer that divides the 48-bit number.
Returns	The 32-bit result of this divide operation.
Description	Divide a 16-bit integer into a 48-bit integer. Expect the answer to be no greater than 32-bits, so return the answer as a 32-bit integer.
File	div48.c
Comments	--

int32_t div_sh48(int16_t x[3], int sh, int16_t y)	
Parameters	*x* - The 48-bit number to be shifted and then divided. The number is represented as a 3-element 16-bit array. *y* - The 16-bit integer that divides the 48-bit number.
Returns	The 32-bit result of this shift-then-divide operation.
Description	Preshift a 48-bit integer upwards by a specified amount. Then divide a 16-bit integer into the shifted 48-bit one. Expect the answer to be no greater than 32-bits, so return the answer as a 32-bit integer.
File	div_sh48.c
Comments	This is a somewhat domain specific divide operation, but pretty useful when handling dot products.

int32_t imul16(int16_t x, int16_t y)	
Parameters	*x* - Multiplicand *y* - Multiplier
Returns	32-bit result
Description	Implements a 16x16->32 2s-complement multiplier. If a hardware multiplier is available it is used. If no hardware multiplier is available, Booth's algorithm is used to directly implement signed multiply in software.
File	imul16.s43
Comments	--

uint16_t isqrt16(uint16_t h)	
Parameters	*h* - 16-bit number to find the square root of.
Returns	16-bit result with the last 8-bits being fractional.
Description	Calculates the square root of a 16-bit number.
File	isqrt16.s43
Comments	--

uint32_t isqrt32(uint32_t h)	
Parameters	*h* - 32-bit number to find the square root of.
Returns	32-bit result with the last 16-bits being fractional.
Description	Calculates the square root of a 32-bit number.
File	isqrt32.s43
Comments	This should not be called with h being a negative number.

uint16_t isqrt32i(uint32_t h)	
Parameters	*h* - 32-bit number to find the square root of.
Returns	Returns the integer portion of the square root of a 32-bit number. This number is rounded to the nearest integer.
Description	Calculates the integer portion (rounded to the nearest integer) of the square root of a 32-bit number.
File	isqrt32i.c
Comments	This should not be called with h being a negative number.

uint64_t isqrt64(uint64_t h)	
Parameters	*h* - 64-bit number to find the square root of.
Returns	64-bit result with the last 32-bits being fractional.
Description	Calculates the square root of a 64-bit number.
File	isqrt64.s43
Comments	This should not be called with h being a negative number.

uint32_t isqrt64i(uint64_t h)	
Parameters	*h* - Number to find the square root of.
Returns	Returns the integer portion of the square root of a 64-bit number. This number is rounded to the nearest integer.
Description	Calculates the integer portion (rounded to the nearest integer) of the square root of a 64-bit number.
File	isqrt64i.c
Comments	This should not be called with h being a negative number.

void mac48_16(int16_t z[3], int16_t x, int16_t y)	
Parameters	*z* - 48-bit number where accumulation takes place. It is represented as a 3-element 16-bit array. *y* - 16-bit multiplicand. *x* - 16-bit multiplier.
Returns	--
Description	Replaces a 48-bit number (z) with the sum of its current value and the product of two 16-bit numbers (x and y); that is, z = z + (x * y).
File	mac48.s43
Comments	This is not protected against interrupts, so only use it in an interrupt routine.

*void mac64_16_24(int64_t *z, int16_t x, int32_t y)*	
Parameters	*z* - Pointer to the 64-bit number where accumulation takes place. *y* - 16-bit multiplicand. *x* - 32-bit multiplier.
Returns	--
Description	Replaces a 64-bit number (z) with the sum of its current value and the product of a 16-bit number (x) and 32-bit number (y); that is, z = z + (x * y) .
File	mac64_16_24.s43
Comments	This is not protected against interrupts, so only use it in an interrupt routine. This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

*static __inline__ void mac64_16_24(int64_t *z, int16_t x, int32_t y)*	
Parameters	*z* - Pointer to the 64-bit number where accumulation takes place. *y* - 16-bit multiplicand. *x* - 32-bit multiplier.
Returns	--
Description	Replaces a 64-bit number (z) with the sum of its current value and the product of a 16-bit number (x) and 32-bit number (y); that is, z = z + (x * y).
File	emeter-toolkit.h
Comments	This is not protected against interrupts, so only use it in an interrupt routine. This version of the function is used for meters with a 32-bit hardware multiplier.

int32_t mul48_32_16(int32_t x, int16_t y)	
Parameters	*x* - Signed 32-bit multiplicand. *y* - Signed 16-bit multiplier.
Returns	Top 32-bits of 48-bit signed result
Description	Multiply a 32-bit signed number (x) by a 16-bit signed number and return the top 32-bits of the 48-bit signed result.
File	mul48_32_16.s43
Comments	This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

static __inline__ int32_t mul48_32_16(int32_t x, int16_t y)	
Parameters	*x* - Signed 32-bit multiplicand. *y* - Signed 16-bit multiplier.
Returns	Top 32-bits of 48-bit signed result.
Description	Multiply a 32-bit signed number (x) by a 16-bit signed number and return the top 32-bits of the 48-bit signed result.
File	emeter-toolkit.h
Comments	This version of the function is used for meters with a 32-bit hardware multiplier.

uint32_t mul48u_32_16(uint32_t x, uint16_t y);	
Parameters	*x* - 32-bit unsigned multiplicand. *y* - 16-bit unsigned multiplier.
Returns	Top 32-bits of 48-bit unsigned result.
Description	Multiply a 32-bit unsigned number (x) by a 16-bit unsigned number and return the top 32-bits of the 48-bit unsigned result.
File	mul48u_32_16.s43
Comments	This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

static __inline__ uint32_t mul48u_32_16(uint32_t x, uint16_t y)	
Parameters	*x* - 32-bit unsigned multiplicand. *y* - 16-bit unsigned multiplier.
Returns	Top 32-bits of 48-bit unsigned result.
Description	Multiply a 32-bit unsigned number (x) by a 16-bit unsigned number and return the top 32-bits of the 48-bit unsigned result.
File	emeter-toolkit.h
Comments	This version of the function is used for meters with a 32-bit hardware multiplier.

int16_t q1_15_mul(int16_t x, int16_t y)	
Parameters	*x* - Multiplicand. *y* - Multiplier.
Returns	Result in Q1.15 format.
Description	16-bit result in Q1.15 style 16x16=>16 multiply.
File	q1_15_mul.s43
Comments	This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

static __inline__ int16_t q1_15_mul(int16_t x, int16_t y)	
Parameters	*x* - Multiplicand. *y* - Multiplier.
Returns	Result in Q1.15 format.
Description	16-bit result in Q1.15 style 16x16=>16 multiply.
File	emeter-toolkit.h
Comments	This version of the function is used for meters with a 32-bit hardware multiplier.

int16_t q1_15_mulr(int16_t x, int16_t y)	
Parameters	*x* - Multiplicand. *y* - Multiplier.
Returns	Result in Q1.15 format with half bit rounding of the result.
Description	16-bit rounded result in Q1.15 style 16x16=>16 multiply.
File	q1_15_mulr.s43
Comments	This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

static __inline__ int16_t q1_15_mulr(int16_t x, int16_t y)	
Parameters	*x* - Multiplicand. *y* - Multiplier.
Returns	Result in Q1.15 format with half bit rounding of the result.
Description	16-bit rounded result in Q1.15 style 16x16=>16 multiply.
File	emeter-toolkit.h
Comments	This version of the function is used for meters with a 32-bit hardware multiplier.

void shift48(register int16_t x[3], int how_far)	
Parameters	*x* - 48-bit number to be shifted. The 48-bit number is represented by a 3-element array of 16-bits. *how_far* - The shift amount. A positive value would shift to the left and a negative value would shift to the right.
Returns	--
Description	Shifts a 48-bit number; that is, (x << how_far).
File	shift48.s43
Comments	--

void sqac48_16(register int16_t z[3], register int16_t x)	
Parameters	*z* - 48-bit number where accumulation takes place. It is represented as a 3-element 16-bit array. *x* - 16-bit number to be squared and added to z.
Returns	--
Description	Replaces a 48-bit number (z) with the sum of its current value with the square of a 16-bit numbers (x); that is, z = z + (x * x).
File	sqac48_16.s43
Comments	This is not protected against interrupts, so only use it in an interrupt routine.

*void sqac64_24(int64_t *z, int32_t x)*	
Parameters	*z* - Pointer to a 64-bit number where accumulation takes place. *x* - 32-bit number to be squared and added to z.
Returns	--
Description	Replaces a 64-bit number (z) with the sum of its current value with the square of a 32-bit numbers (x); that is, z= z + (x * x).
File	sqac64_24.s43
Comments	This is not protected against interrupts, so only use it in an interrupt routine. This version of the function is used for meters with either 16-bit hardware multipliers or no hardware multipliers at all.

*static __inline__ void sqac64_24(int64_t *z, int32_t x)*	
Parameters	*z* - Pointer to a 64-bit number where accumulation takes place. *x* - 32-bit number to be squared and added to z.
Returns	--
Description	Replaces a 64-bit number (z) with the sum of its current value with the square of a 32-bit numbers (x); that is, z= z + (x * x).
File	emeter-toolkit.h
Comments	This is not protected against interrupts, so only use it in an interrupt routine. This version of the function is used for meters with a 32-bit hardware multiplier.

2.2 Metrology Project

*int trng(uint16_t *val)*	
Parameters	*val* - Pointer to variable where the random number is to be stored.
Returns	1 if a new random number is not available. 0 if a new random number is available.
Description	Get a random number, if available, from the true random number generator that is based on Gaussian noise in the LSB of the thermal diode. The random value is stored in val.
File	emeter-background.c
Comments	This function only available if the temperature is being measured, which is disabled for the AFE253 code to lower amount of RAM used.

uint16_t trng_wait(void)	
Parameters	--
Returns	A random number
Description	Get a random number from the true random number generator, based on Gaussian noise in the LSB of the thermal diode. If a random number is not available, the function waits until it is available.
File	emeter-background.c
Comments	This function is available only if the temperature is being measured, which is disabled for the AFE253 code to reduce the amount of RAM that is used.

static void __inline__ log_parameters(void)	
Parameters	--
Returns	--
Description	Takes a snapshot of various values for logging purposes, clears the working values so data can be captured for the next analysis period, and then tells the main function to deal with the snapshot values by asserting the NEW_LOG flag.
File	emeter-background.c
Comments	This function is for single-phase meters only. The values to be logged are stored in the phase structure. Multiphase meters have a version of this function that takes a pointer to a structure that has the working data of the phase to be logged.

*static void __inline__ log_parameters(struct phase_parms_s *phase)*	
Parameters	*phase* - Pointer to a struct that contains the working parameters of the current phase.
Returns	--
Description	Takes a snapshot of various values for logging purposes, clears the working values so data can be captured for the next analysis period, and then tells the main function to deal with the snapshot values by asserting the NEW_LOG flag.
File	emeter-background.c
Comments	This function is for multi-phase meters. The single phase version of this function that has no input parameters since it assumes the values to be logged are in the phase structure.

static void __inline__ log_neutral_parameters(void)	
Parameters	--
Returns	--
Description	Logs neutral lead information for multi-phase meters.
File	emeter-background.c
Comments	This function is not available for the MSP430AFE, because it available for single-phase meters or meters without neutral monitoring support.

void adc_interrupt(void)	
Parameters	--
Returns	--
Description	Interrupt routine where the main signal processing is done. In this routine, current and voltage samples are obtained. Each of these samples are then squared and accumulated. Also, voltage and current samples are multiplied together to calculate instantaneous active power. The instantaneous power is accumulated until it reaches a user-specified threshold, at which time, a pulse is outputted. In this routine, the necessary data is obtained so that frequency, thd parameters, reactive power, apparent power, random numbers, and other parameters can be calculated later.
File	emeter-background.c
Comments	When about 1 second of samples has been obtained and processed, the log_parameters, (and if applicable) log_neutral_parameters functions are called.

void limp_trigger_interrupt(void)	
Parameters	--
Returns	--
Description	Interrupt routine to trigger the Sigma Delta ADCs when running in limp mode.
File	emeter-background.c
Comments	This is available only for single-phase meters that support limp mode and have a Sigma Delta converter.

void adc10_interrupt (void)	
Parameters	--
Returns	--
Description	Interrupt routine to handle the ADC10A in the 6xx family devices
File	emeter-background.c
Comments	--

*void set_phase_correction(struct phase_correction_s *s, int correction)*	
Parameters	*s* - The phase correction structure to be updated to help to produce the proper delay. *correction* - The correction amount where the lower 8-bits are used as indices into arrays to calculate the fir gain and beta. The other bits are used to correspond to number of sample delays.
Returns	--
Description	Finds the proper delay. This is used in particular to calculate the 90° shifted voltage samples.
File	emeter-foreground.c
Comments	--

*static void set_phase_gain_correction(struct phase_correction_s *s, int correction, int gain)*	
Parameters	*s* - The phase correction structure to be updated. *correction* - The correction amount where the lower 8-bits are used as indices into arrays to calculate the fir gain and beta. The other bits are used to correspond to number of sample delays. *gain* - Constant used to q1.15 multiply the fir gain value obtained from using correction to index into the fir gain table.
Returns	--
Description	This function is used in dynamic phased correction.
File	emeter-foreground.c
Comments	--

*void set_sd_phase_correction(struct phase_correction_sd16_s *s, int ph, int correction)*	
Parameters	*s* - The phase correction structure to be updated. *ph*- Which phase is to be corrected. *correction* - The correction amount where the lower 8-bits correspond to delay that would be applied to the preload register. The other bits are used to correspond to number of sample delays.
Returns	--
Description	Performs phase correction for Sigma Delta ADCs.
File	emeter-foreground.c
Comments	This function is not available for the MSP430AFE, because it available if DYNAMIC_PHASE_CORRECTION_SUPPORT is not defined.

static int32_t test_phase_balance(int32_t live_signal, int32_t neutral_signal, int threshold)	
Parameters	*Live_signal* - The value of the live signal. This could either correspond to current or power. *Neutral_signal* - The value of the neutral signal. This could either correspond to current or power. *Threshold* - Value at which if both the live and neutral signals are below, a relaxed balanced fraction would be used for determining phase unbalance.
Returns	Returns the signal with the highest value between live_signal and neutral_signal.
Description	Tests between two currents, or between two powers. In normal mode it is testing between two power readings. In limp mode it is testing between two current readings. The function sees which signal (live or neutral) is bigger, with some tolerance built in. If the signal measured from the neutral is more than 6.25% or 12.5% (options) different from the signal measured from the live there is something wrong (maybe fraudulent tampering, or just something faulty). In this case, the current measured from the channel with the higher signal is used. When the channel is reasonably balanced, use the signal from the live lead. If neither signal is above the threshold, use a more relaxed measure of imbalance (say 25% or even 50%), to allow for the lower accuracy of these small. Assessments are persistence checked to avoid transient conditions causing a false change of imbalance status.
File	emeter-foreground.c
Comments	This function is available only if NEUTRAL_MONITOR_SUPPORT and POWER_BALANCE_DETECTION_SUPPORT are both defined.

int16_t frequency(void)	
Parameters	--
Returns	The measured frequency in 0.01 Hz resolution.
Description	Uses the parameters calculated from the background process, to calculate the frequency.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if MAINS_FREQUENCY_SUPPORT is defined.

*int16_t frequency(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured frequency for the desired phase in .01 Hz resolution.
Description	Uses the parameters calculated from the background process to calculate the frequency.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if MAINS_FREQUENCY_SUPPORT is defined.

rms_voltage_t voltage(void)	
Parameters	--
Returns	The measured RMS voltage in 1 mV resolution.
Description	Uses the parameters calculated from the background process to calculate the RMS voltage.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if VRMS_SUPPORT is defined.

*rms_voltage_t voltage(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured RMS voltage for the desired phase in 1 mV resolution.
Description	Uses the parameters calculated from the background process to calculate the RMS voltage.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if VRMS_SUPPORT is defined.

rms_voltage_t fundamental_voltage(void)	
Parameters	--
Returns	The measured fundamental voltage in 1 mV resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental voltage.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if FUNDAMENTAL_VRMS_SUPPORT is defined.

*rms_voltage_t fundamental_voltage(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured fundamental voltage for the desired phase in 1 mV resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental voltage.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if FUNDAMENTAL_VRMS_SUPPORT is defined.

int16_t voltage_thd(void)	
Parameters	--
Returns	The THD of the voltage waveform.
Description	Uses the parameters calculated from the background process to calculate the thd percentage of the voltage waveform.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if VOLTAGE_THD_SUPPORT is defined.

*int16_t voltage_thd(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The THD percentage of the voltage waveform.
Description	Uses the parameters calculated from the background process to calculate the thd of the voltage waveform.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if VOLTAGE_THD_SUPPORT is defined.

*void dynamic_phase_correction(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv, int ph)*	
Parameters	*phase* - Pointer to the structure that has the parameters that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values.*ph* - The phase number.
Returns	--
Description	Performs dynamic phase correction.
File	emeter-foreground.c
Comments	This function is available only if DYNAMIC_PHASE_CORRECTION_SUPPORT is defined.

*void dynamic_phase_correction_neutral(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv, int ph)*	
Parameters	*phase* - Pointer to the structure that has the parameters that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values.*ph* - The phase number.
Returns	--
Description	Performs dynamic phase correction for the neutral channel.
File	emeter-foreground.c
Comments	This function is available only for single-phase meters. This function is also available only if NEUTRAL_MONITOR_SUPPORT and DYNAMIC_PHASE_CORRECTION_SUPPORT are defined.

rms_current_t current(void)	
Parameters	--
Returns	The measured RMS current in 1 µA resolution.
Description	Uses the parameters calculated from the background processW to calculate the RMS current.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if IRMS_SUPPORT is defined.

*rms_current_t current(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv, int ph)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase. *ph* - The phase number.
Returns	The measured RMS current for the desired phase in 1 µA resolution.
Description	Uses the parameters calculated from the background processW to calculate the RMS current.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if IRMS_SUPPORT is defined.

rms_current_t fundamental_current(void)	
Parameters	--
Returns	The measured fundamental current in .1 mA resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental current.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if FUNDAMENTAL_IRMS_SUPPORT is defined.

*rms_current_t fundamental_current(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured fundamental current for the desired phase in .1 mA resolution.
Description	Uses the parameters calculated from the background process, to calculate the fundamental current.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if FUNDAMENTAL_IRMS_SUPPORT is defined.

int16_t current_thd(void)	
Parameters	--
Returns	The THD percentage of the current waveform.
Description	Uses the parameters calculated from the background process to calculate the thd of the current waveform.
File	emeter-foreground.c
Comments	This function is for single-phase meters. This function is available only if CURRENT_THD_SUPPORT is defined.

*int16_t current_thd(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The THD percentage of the current waveform.
Description	Uses the parameters calculated from the background process to calculate the thd of the current waveform.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if CURRENT_THD_SUPPORT is defined.

rms_current_t neutral_current(void)	
Parameters	--
Returns	The neutral channel's measured RMS current.
Description	Uses the parameters calculated from the background process,to calculate the neutral channel's RMS current.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if both IRMS_SUPPORT and NEUTRAL_MONITOR_SUPPORT are defined.

rms_current_t residual_current(void)	
Parameters	--
Returns	The residual current.
Description	Uses the parameters calculated from the background process to calculate the residual current.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. This function is available only if both RESIDUAL_IRMS_SUPPORT and NEUTRAL_MONITOR_SUPPORT are defined.

power_t active_power(void)	
Parameters	--
Returns	The measured active power in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the active power.
File	emeter-foreground.c
Comments	This function is for single-phase meters.

*power_t active_power(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured active power for the desired phase in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the active power.
File	emeter-foreground.c
Comments	This function is for multi-phase meters.

power_t reactive_power(void)	
Parameters	--
Returns	The measured reactive power in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the reactive power.
File	emeter-foreground.c
Comments	This function is for single-phase meters. For this function to be available, REACTIVE_POWER_SUPPORT and REACTIVE_POWER_BY_QUADRATURE_SUPPORT must be defined.

*power_t reactive_power(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured reactive power for the desired phase in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the reactive power.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. For this function to be available, REACTIVE_POWER_SUPPORT and REACTIVE_POWER_BY_QUADRATURE_SUPPORT must be defined.

int32_t apparent_power(void)	
Parameters	--
Returns	The measured apparent power in 10 mW resolution.
Description	Uses the parameters calculated from the background process, to calculate the apparent power.
File	emeter-foreground.c
Comments	This function is for single-phase meters. For this function to be available, APPARENT_POWER_SUPPORT must be defined.

*int32_t apparent_power(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured apparent power for the desired phase in 10 mW resolution.
Description	Uses the parameters calculated from the background process, to calculate the apparent power.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. For this function to be available, APPARENT_POWER_SUPPORT must be defined.

power_t fundamental_active_power(void)	
Parameters	--
Returns	The measured fundamental active power in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental active power.
File	emeter-foreground.c
Comments	This function is for single-phase meters. For this function to be available, FUNDAMENTAL_ACTIVE_POWER_SUPPORT must be defined.

*power_t fundamental_active_power(struct phase_parms_s *phase, struct sphase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured fundamental active power for the desired phase in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental active power.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. For this function to be available, FUNDAMENTAL_ACTIVE_POWER_SUPPORT must be defined.

power_t fundamental_reactive_power(void)	
Parameters	--
Returns	The measured fundamental reactive power in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental reactive power.
File	emeter-foreground.c
Comments	This function is for single-phase meters. For this function to be available, FUNDAMENTAL_REACTIVE_POWER_SUPPORT must be defined.

*power_t fundamental_reactive_power(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured fundamental reactive power for the desired phase in 10 mW resolution.
Description	Uses the parameters calculated from the background process to calculate the fundamental reactive power.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. For this function to be available, FUNDAMENTAL_REACTIVE_POWER_SUPPORT must be defined.

int16_t power_factor(void)	
Parameters	--
Returns	The measured power factor.
Description	Uses the parameters calculated from the background process to calculate the power factor.
File	emeter-foreground.c
Comments	This function is for single-phase meters. For this function to be available, POWER_FACTOR_SUPPORT must be defined.

*int16_t power_factor(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The measured power factor.
Description	Uses the parameters calculated from the background process to calculate the power factor for the desired phase.
File	emeter-foreground.c
Comments	This function is for multi-phase meters. For this function to be available, POWER_FACTOR_SUPPORT must be defined.

void temperature(void)	
Parameters	--
Returns	--
Description	Find the temperature in Celsius and update the temperature_in_celsius global variable to that temperature.
File	emeter-foreground.c
Comments	For this function to be available, TEMPERATURE_SUPPORT must be defined.

power_t calculate_readings(void)	
Parameters	--
Returns	The active power.
Description	Calculate the metering parameters by calling the individual functions that calculate these parameters. The individual functions make the parameter calculations using the data from the background process.
File	emeter-foreground.c
Comments	This function is for single-phase meters.

*power_t calculate_readings(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv, int ch)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase. *ch* - Channel number.
Returns	The measured active power of the desired phase.
Description	Calculate the metering parameters of the desired phase by calling the individual functions that calculate these parameters. The individual functions make the parameter calculations using the data from the background process.
File	emeter-foreground.c
Comments	This function is for multi-phase meters.

power_t calculate_limp_readings(void)	
Parameters	--
Returns	The limp-mode active power value that was calculated using the nominal voltage.
Description	Calculate the limp-mode metering parameters by calling the individual functions that calculate these parameters. The individual functions make the parameter calculations using the data from the background process.
File	emeter-foreground.c
Comments	This function is for single-phase meters.

*power_t calculate_limp_readings(struct phase_parms_s *phase, struct phase_nv_parms_s const *phase_nv)*	
Parameters	*phase* - Pointer to the structure that has the parameters for the desired phase that was calculated by the background process. *phase_nv* - Pointer to the structure that has the calibration values for the desired phase.
Returns	The limp-mode active power for the desired phase.
Description	Calculate the limp-mode metering parameters of the desired phase by calling the individual functions that calculate these parameters. The individual functions make the parameter calculations using the data from the background process.
File	emeter-foreground.c
Comments	This function is for multi-phase meters.

void calculate_neutral_readings(void)	
Parameters	--
Returns	--
Description	Calculate the neutral metering parameters by calling the individual functions that calculate these parameters. The individual functions make the parameter calculations using the data from the background process.
File	emeter-foreground.c
Comments	This function is for multi-phase meters.

void metrology_limp_normal_detection(void)	
Parameters	--
Returns	--
Description	Detect when the meter should enter limp mode from normal mode and when to go back to normal mode from limp mode. Calls the necessary functions to switch modes.
File	emeter-foreground.c
Comments	For this function to be available, LIMP_MODE_SUPPORT must be defined.

void metrology_init_analog_front_end_normal_mode(void)	
Parameters	--
Returns	--
Description	Configures the sigma-delta ADC module as an analog front-end for a meter that is running in normal mode.
File	emeter-metrology-setup.c
Comments	--

void metrology_init_analog_front_end_limp_mode(void)	
Parameters	--
Returns	--
Description	Configures the sigma-delta ADC module as an analog front-end for a meter that is running in limp mode.
File	emeter-metrology-setup.c
Comments	This function is available only if limp mode is supported by the meter.

void metrology_switch_to_powerfail_mode(void)	
Parameters	--
Returns	--
Description	Configures the meter to run in powerfail mode when a power failure has occurred. When power is restored, return the meter back to limp mode (if supported) or normal mode.
File	emeter-metrology-setup.c
Comments	This function is available only if POWER_DOWN_SUPPORT is supported by the meter.

static __inline__ int64_t int48_to_64(int16_t x[3])	
Parameters	*x* - 48-bit number to be converted to 64-bit. The 48-bit number is represented as a 3-element array of 16 bits.
Returns	64-bit integer representation of the input 48-bit number.
Description	Converts a 48-bit int into a 64-bit int.
File	emeter-toolkit.h
Comments	--

static __inline__ void int64_to_48(int16_t y[3], int64_t x)	
Parameters	*x* - 64-bit number to be converted to 48-bit. The 48-bit number is represented as a 3-element array of 16-bits.
Returns	48-bit integer representation of the input 64-bit number. The top 2 bytes of the 64-bit number are disregarded.
Description	Converts a 64-bit int into a 48-bit int.
File	emeter-toolkit.h
Comments	--

static __inline__ void transfer48(int16_t y[3], int16_t x[3])	
Parameters	*x* - Source represented as a 48-bit, 3-element 16-bit array. *y* - Destination represented as a 48-bit, 3-element 16 array.
Returns	--
Description	Transfers a 48-bit variable to another 48-bit variable. The source variable is then set to zero.
File	emeter-toolkit.h
Comments	--

static __inline__ void assign48(int16_t y[3], const int16_t x[3])	
Parameters	*x* - Source represented as a 48-bit, 3-element 16-bit array. *y* - Destination represented as a 48-bit, 3-element 16 array.
Returns	--
Description	Transfers a 48-bit variable to another 48-bit variable.
File	emeter-toolkit.h
Comments	--

static __inline__ int16_t ADC16_0(void)	
Parameters	--
Returns	Conversion value of channel 0 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 0 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ int16_t ADC16_0(void)	
Parameters	--
Returns	Returns the SD16CCTL0 flag. If the flag is zero, then a conversion has not occurred for channel 0 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 0 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ void ADC16_0_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL0 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ int16_t ADC16_1(void)	
Parameters	--
Returns	Conversion value of channel 1 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 1 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ int16_t ADC16_1(void)	
Parameters	--
Returns	Returns the SD16CCTL1 flag. If the flag is zero, then a conversion has not occurred for channel 1 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 1 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ void ADC16_1_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL1 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default.

static __inline__ int16_t ADC16_2(void)	
Parameters	--
Returns	Conversion value of channel 2 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 2 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ int16_t ADC16_2(void)	
Parameters	--
Returns	Returns the SD16CCTL2 flag. If the flag is zero, then a conversion has not occurred for channel 2 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 2 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ void ADC16_2_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL2 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ int16_t ADC16_3(void)	
Parameters	--
Returns	Conversion value of channel 3 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 1 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ int16_t ADC16_3(void)	
Parameters	--
Returns	Returns the SD16CCTL3 flag. If the flag is zero, then a conversion has not occurred for channel 3 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 3 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ void ADC16_3_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL3 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ int16_t ADC16_4(void)	
Parameters	--
Returns	Conversion value of channel 4 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 4 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ int16_t ADC16_4(void)	
Parameters	--
Returns	Returns the SD16CCTL4 flag. If the flag is zero, then a conversion has not occurred for channel 4 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 4 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ void ADC16_4_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL4 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ int16_t ADC16_5(void)	
Parameters	--
Returns	Conversion value of channel 5 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 5 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ int16_t ADC16_5(void)	
Parameters	--
Returns	Returns the SD16CCTL5 flag. If the flag is zero, then a conversion has not occurred for channel 5 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 5 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ void ADC16_5_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL5 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ int16_t ADC16_6(void)	
Parameters	--
Returns	Conversion value of channel 6 of the ADC when it is running in 16-bit mode.
Description	Gets the conversion result of channel 6 of the ADC when running in 16-bit mode.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

static __inline__ int16_t ADC16_6(void)	
Parameters	--
Returns	Returns the SD16CCTL6 flag. If the flag is zero, then a conversion has not occurred for channel 6 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 6 has completed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

static __inline__ void ADC16_6_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL6 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_16bit_access.h
Comments	In the emeter library, voltages are set to run in 16-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

static __inline__ int32_t ADC32_0(void)	
Parameters	--
Returns	Conversion value of channel 0 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 0 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ int32_t ADC32_0(void)	
Parameters	--
Returns	Returns the SD16CCTL0 flag. If the flag is zero, then a conversion has not occurred for channel 0 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 0 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ void ADC32_0_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL0 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ int32_t ADC32_1(void)	
Parameters	--
Returns	Conversion value of channel 1 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 1 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ int32_t ADC32_1(void)	
Parameters	--
Returns	Returns the SD16CCTL1 flag. If the flag is zero, then a conversion has not occurred for channel 1 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 1 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ void ADC32_1_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL1 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default.

static __inline__ int32_t ADC32_2(void)	
Parameters	--
Returns	Conversion value of channel 2 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 2 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ int32_t ADC32_2(void)	
Parameters	--
Returns	Returns the SD16CCTL2 flag. If the flag is zero, then a conversion has not occurred for channel 2 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 2 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ void ADC32_2_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL2 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has three or more sigma delta converters.

static __inline__ int32_t ADC32_3(void)	
Parameters	--
Returns	Conversion value of channel 3 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 3 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ int32_t ADC32_3(void)	
Parameters	--
Returns	Returns the SD16CCTL3 flag. If the flag is zero, then a conversion has not occurred for channel 3 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 3 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ void ADC32_3_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL3 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has four or more sigma delta converters.

static __inline__ int32_t ADC32_4(void)	
Parameters	--
Returns	Conversion value of channel 4 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 4 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ int32_t ADC32_4(void)	
Parameters	--
Returns	Returns the SD16CCTL4 flag. If the flag is zero, then a conversion has not occurred for channel 4 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 4 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ void ADC32_4_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL4 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has five or more sigma delta converters.

static __inline__ int32_t ADC32_5(void)	
Parameters	--
Returns	Conversion value of channel 5 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 5 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ int32_t ADC32_5(void)	
Parameters	--
Returns	Returns the SD16CCTL5 flag. If the flag is zero, then a conversion has not occurred for channel 5 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 5 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ void ADC32_5_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL5 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has six or more sigma delta converters.

static __inline__ int32_t ADC32_6(void)	
Parameters	--
Returns	Conversion value of channel 6 of the ADC when it is running in 24-bit mode.
Description	Gets the conversion result of channel 6 of the ADC when running in 24-bit mode.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

static __inline__ int32_t ADC32_6(void)	
Parameters	--
Returns	Returns the SD16CCTL6 flag. If the flag is zero, then a conversion has not occurred for channel 6 since the last time this flag was cleared. If the value is nonzero, then a new conversion has completed.
Description	Determines if a new conversion for channel 6 has completed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

static __inline__ void ADC32_6_CLEAR	
Parameters	--
Returns	--
Description	Clears the SD16CCTL6 flag. This is used to distinguish new conversion results from old conversion results and to prevent the ADC interrupt from being retriggered after a conversion result has been processed.
File	sigma_delta_24bit_access.h
Comments	In the emeter library, currents are set to run in 24-bit mode by default. This function is available only if the meter has seven or more sigma delta converters.

2.3 Application Project

*void lcd_text(char *s, int pos)*	
Parameters	*s* - Null terminated Ascii string. *pos* - LCD position where text should start to be displayed, where 1 is the most left-most character.
Returns	--
Description	Displays an ascii string on a LCD.
File	emeter-basic-display.c
Comments	This function is only used for LCDs that support displaying ascii characters. This function is not available for the MSP430AFE, because it does not have a LCD driver.

*void LCDcharsx(const lcd_cell_t *s, int pos, int len)*	
Parameters	*s* - An array of character code values needed to produce the desired string on the LCD. For example, the first element in this array corresponds to the character code for the first character to be displayed on the LCD. *pos* - LCD position where text should start to be displayed, where 1 is the most left-most character. *len*-Number of characters of the string to display on the LCD.
Returns	--
Description	Displays characters on a LCD.
File	emeter-basic-display.c
Comments	This function has different versions depending on whether a starburst display is used. This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_power_fail_message(void)	
Parameters	--
Returns	--
Description	Displays "bl out" on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_startup_message(void)	
Parameters	--
Returns	--
Description	Displays "START" on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_power_4v2_message(void)	
Parameters	--
Returns	--
Description	Displays "4V2" or "4U2" on the LCD, depending on the actual LCD display type.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_power_normal_message(void)	
Parameters	--
Returns	--
Description	Displays "8V4" or "8U4" on the LCD, depending on the actual LCD display type.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static void LCDicon(int pos, int on)	
Parameters	*pos* - An integer that determines the proper LCD memory register and bit for each symbol. This value can be calculated by using the icon_loc(cell,bit) macro. *on* - Turns a character on if this parameter is 1. Turns a character off if this parameter is 0.
Returns	--
Description	Turn a LCD icon on or off.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_clear_periphery(void)	
Parameters	--
Returns	--
Description	Clear all the symbols around the display, which are not being used.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_clear_line_1(void)	
Parameters	--
Returns	--
Description	Clear the first line of the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_clear_line_2(void)	
Parameters	--
Returns	--
Description	Clear the second line of the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver. This function is also available only for LCDs with two lines. Calling this function for LCDs that only have one line would clean line 1.

void display_phase_icon(int ph)	
Parameters	*ph* - The phase whose symbol is to be displayed. A value of 0 corresponds to phase a, 1 to phase b, and 2 for phase c.
Returns	--
Description	Displays the icon that signifies the current phase whose parameters are being displayed on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver. This function is also available only for multiphase meters that have symbols to signify which phase the currently displayed parameters belong to.

static void LCDoverrange1(void)	
Parameters	--
Returns	--
Description	Displays "High" on the LCD's first line.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static void LCDoverrange2(void)	
Parameters	--
Returns	--
Description	Displays "High" on the LCD's second line. If the LCD does not have a second line, it displays it on the first line.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_mains_frequency(int ph)	
Parameters	*ph* - Phase number to display the frequency of.
Returns	--
Description	Displays frequency of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static void display_vrms(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays root mean square voltage of the selected phase on the LCD .
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_irms(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays root mean square current of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_consumed_active_energy(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays consumed active energy of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_consumed_reactive_energy(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays consumed reactive energy of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_active_power(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays active power of the selected phase on the LCD
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_reactive_power(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays reactive power of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_apparent_power(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays apparent power of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_power_factor(int ph)	
Parameters	*ph* - Phase number.
Returns	--
Description	Displays power factor of the selected phase on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static void display_date(int year, int month, int day)	
Parameters	*year* - Year to be displayed. *month* - Month to be displayed. *day* - Day to be displayed.
Returns	--
Description	Displays a date on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static void display_time(int hour, int minute, int second)	
Parameters	*hour* - Hour to be displayed. *minute*- Minute to be displayed. *second* - Second to be displayed.
Returns	--
Description	Displays a time on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static / void display_current_date(void)	
Parameters	--
Returns	--
Description	Displays current date on the LCD
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

static __inline__ void display_current_time(void)	
Parameters	--
Returns	--
Description	Displays current time on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_current_tariff(void)	
Parameters	--
Returns	--
Description	Displays the current tariff on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_tariff_holiday(void)	
Parameters	--
Returns	--
Description	Displays the dates of all holidays on the LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void display_item(int item, int ph)	
Parameters	*item* - Integer that determines what parameter to display on the LCD. *ph* - Which phase's parameter to display.
Returns	--
Description	Displays on the LCD the specified parameter of the specified phase.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void update_display(void)	
Parameters	--
Returns	--
Description	Update the LCD to cycle through displaying different parameters for each phase.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void send_1107d_report(void)	
Parameters	--
Returns	--
Description	Send 1107d report via Com port.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*int iec62056_21_process_ack(const uint8_t *msg, int len)*	
Parameters	--
Returns	--
Description	Process ack.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*int iec62056_21_process_nak(const uint8_t *msg, int len)*	
Parameters	--
Returns	--
Description	Process nack
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*int iec62056_21_process_ident(const uint8_t *msg, int len)*	
Parameters	--
Returns	--
Description	Process ident.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void iec62056_21_process_request(void)	
Parameters	--
Returns	--
Description	Process request.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*int iec62056_21_process_header(const uint8_t *msg, int len)*	
Parameters	--
Returns	--
Description	Process header.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*int iec62056_21_process_field(const uint8_t instance_id[6], const uint8_t *val, int len)*	
Parameters	--
Returns	--
Description	Process field.
File	emeter-communication.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void serial_rx_interrupt0 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for receiving data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has UART_0 or USCI_AB0.

void serial_tx_interrupt0 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has UART_0 or USCI_AB0.

void serial_interrupt0 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for receiving and sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has a USCI_A0 or EUSCI_A0.

void serial_rx_interrupt1 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for receiving data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has UART_1 or USCI_AB1.

void serial_tx_interrupt1 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has UART_1 or USCI_AB1.

void serial_interrupt1 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for receiving and sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has a USCI_A1 or EUSCI_A1.

void serial_tx_interrupt2 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has USCI_AB2.

void serial_tx_interrupt2 (void)	
Parameters	--
Returns	--
Description	Interrupt routine for sending data via the Com port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has USCI_AB3.

void send_message(int port, int len)	
Parameters	*port* - Port number whose tx interrupt is desired to be triggered. *len* - Length of the message to be sent.
Returns	--
Description	Triggers the tx interrupt of the specified port.
File	emeter-communication.c
Comments	This function is defined only if the MSP430 has USCI_AB3.

void comms_setup(void)	
Parameters	--
Returns	--
Description	Configures the com ports for communication.
File	emeter-communication.c
Comments	--

int prepare_tx_message(int port, int len)	
Parameters	*port* - Port number whose tx interrupt is desired to be triggered. *len* - Length of the message to be sent.
Returns	--
Description	Formats a transmit message to adhere to dlt645 format. The tx interrupt is then triggered to send the message.
File	emeter-dlt645.c
Comments	--

*static void dlt645_process_rx_message(int port, serial_msg_t *rx_msg, int rx_len)*	
Parameters	*port* - Port number where dlt645 message came from. *rx_msg* - Pointer to the received message. *rx_len* - Length of the received message.
Returns	--
Description	Interpret received dlt645 messages and take the proper actions.
File	emeter-dlt645.c
Comments	

void dlt645_rx_byte(int port, uint8_t ch)	
Parameters	*port* - Port number where received byte came from. *len* - Length of the message to be sent.
Returns	--
Description	Called when receive a byte. Determine if have received a full dlt645 message and call the function to manage a dlt645 message.
File	emeter-dlt645.c
Comments	--

void dlt645_rx_byte(int port, uint8_t ch)	
Parameters	*port* - Port number where received byte came from. *len* - Length of the message to be sent.
Returns	--
Description	Called when receive a byte. Determine if have received a full dlt645 message and call the function to manage a dlt645 message.
File	emeter-dlt645.c
Comments	--

void LCDinit(void)	
Parameters	--
Returns	--
Description	Initialize the LCD display, and set it to initially display all segments.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDsleep(void)	
Parameters	--
Returns	--
Description	Turn off LCD timing generator. This function is used when the meter goes to sleep.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDawaken(void)	
Parameters	--
Returns	--
Description	Turn on LCD timing generator. This function is used when the meter wakes up from sleeping.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

*void LCDchars(const uint8_t *s, int pos, int len)*	
Parameters	*s* - An array of character code-values needed to produce the desired string on the LCD. For example, the first element in this array corresponds to the character code for the first character to be displayed on the LCD. *pos* - LCD position where text should start to be displayed, where 1 is the most left-most character. *len* - Number of characters of the string to display on the LCD.
Returns	--
Description	Displays characters on a LCD.
File	emeter-basic-display.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDmodify_char(uint16_t ch, int pos, int on)	
Parameters	*pos* - An integer that determines the proper LCD memory register. *ch* - The character code needed to turn on a symbol. *on* - Turns a character on if this parameter is 1. Turns a character off if this parameter is 0.
Returns	--
Description	Turn a LCD icon on or off.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDdecu16(uint16_t value, int pos, int digits, int after)	
Parameters	*value* - Value to be displayed on the LCD. *pos* - An integer that determines the starting location of where the number would be displayed. *digits* - Number to display on the LCD. *after* - The number of digits which are after the decimal point.
Returns	--
Description	Display an unsigned 16-bit integer, with leading zero suppression.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDdecu32(uint32_t value, int pos, int digits, int after)	
Parameters	*value* - Value to be displayed on the LCD. *pos* - An integer that determines the starting location of where the number would be displayed. *digits* - Number to display on the LCD. *after* - The number of digits which are after the decimal point.
Returns	--
Description	Display an unsigned 32-bit integer, with leading zero suppression.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDdec16(int16_t value, int pos, int digits, int after)	
Parameters	*value* - Value to be displayed on the LCD. *pos* - An integer that determines the starting location of where the number would be displayed. *digits* - Number to display on the LCD. *after* - The number of digits which are after the decimal point.
Returns	--
Description	Display a signed 16-bit integer, with leading zero suppression.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void LCDdec32(int32_t value, int pos, int digits, int after)	
Parameters	*value* - Value to be displayed on the LCD. *pos* - An integer that determines the starting location of where the number would be displayed. *digits* - Number to display on the LCD. *after* - The number of digits which are after the decimal point.
Returns	--
Description	Display a signed 32-bit integer, with leading zero suppression.
File	emeter-lcd.c
Comments	This function is not available for the MSP430AFE, because it does not have a LCD driver.

void set_rtc_sumcheck(void)	
Parameters	--
Returns	--
Description	Update the sumcheck of the rtc based on the time and day paramters.
File	emeter-rtc.c
Comments	--

int bump_rtc(void)	
Parameters	--

Returns	An integer that represents an inconsistent RTC if the current sumcheck does not equal the sumcheck stored in the rtc structure or an integer that represents the greatest unit of time (second, minute, hour, day, month, or year) that changed by updating the RTC. For example, if updating the RTC caused the month to change (which also means the day, hour, minute, and second changed), return an identifier to signify only the month change instead of the other parameters changing. The possible values to be returned by this function are:
	RTC_INCONSISTENT = 0
	RTC_CHANGED_SECOND = 1
	RTC_CHANGED_MINUTE = 2
	RTC_CHANGED_HOUR = 3
	RTC_CHANGED_DAY = 4
	RTC_CHANGED_MONTH = 5
	RTC_CHANGED_YEAR = 6

Description	Checks the RTC consistentcy. If the RTC is consistent, update the time by 1 second and update the minute, hour, day, month, and year fields, if necessary. After updating the RTC, return the greatest unit-of-time change that occurred by updating the RTC by one second.
File	emeter-rtc.c
Comments	--

int check_rtc_sumcheck(void))	
Parameters	--
Returns	1 if the RTC is consistent 0 if the RTC is not consistent.
Description	Checks if the RTC is consistent by comparing the stored sumcheck with the sumcheck created by using the current time and day.
File	emeter-rtc.c
Comments	--

int weekday(void)	
Parameters	--
Returns	Day of the week as a number from 0 (Sunday) to 6 (Saturday)
Description	Finds the current day of the week.
File	emeter-rtc.c
Comments	--

void rtc_bumper(void)	
Parameters	--
Returns	--
Description	Call the function to update the RTC time/date (bump_rtc) and the necessary functions corresponding to any time/date parameter change as a result of updating the RTC.
File	emeter-rtc.c
Comments	This function is called in an ISR routine that is triggered every second. This function is called only if RTC_SUPPORT is defined.

void correct_rtc(void)	
Parameters	--
Returns	--
Description	Correct the RTC to allow for basic error in the crystal, and temperature dependant changes. This is called every two seconds, so it must accumulate two seconds worth of error at the current temperature.
File	emeter-rtc.c
Comments	This is currently disabled in the code library.

void one_second_ticker (void)	
Parameters	--
Returns	--
Description	ISR that is triggered once a second. This also handles sensing when to change operating mode from powerfail mode.
File	emeter-rtc.c
Comments	This is currently disabled in the code library.

int32_t assess_rtc_speed(void)	
Parameters	--
Returns	The RTC speed in terms of SMCLK clock cycles.
Description	Can be used to measure the speed difference between the MSP430's crystal and the external clock in a reasonable time
File	emeter-rtc.c
Comments	SMCLK must be running much faster than the rtc in order to get an accurate reading. This function is available only if CORRECTED_RTC_SUPPORT and __MSP430_HAS_TA3__ are defined.

void rtc_init(void)	
Parameters	--
Returns	--
Description	Initialize the rtc structure with a date and time. As of this writing, the code initializes the date to October 9, 2011 at 12:00:00.
File	emeter-rtc.c
Comments	--

int align_hardware_with_calibration_data(void)	
Parameters	--
Returns	0
Description	When calibrating, calibration constants in flash are updated so that the meter can run accurately. This function reinitializes the sigma delta ADCs to work using the new calibration constants.
File	emeter-setup.c
Comments	This function is available only if ESP_SUPPORT is not defined.
void system_setup(void)	
Parameters	--
Returns	--
Description	Initializes the hardware and metering variables.
File	emeter-setup.c
Comments	--

*void flash_clr(int *ptr)*	
Parameters	*ptr* - Address of the integer in flash that is to be cleared.
Returns	--
Description	Clears an integer that is stored in flash.
File	emeter-flash.c
Comments	--

*void flash_write_int8(int8_t *ptr, int8_t value)*	
Parameters	*ptr* - Address of the integer in flash that is to be rewritten. *value* - Value to be written to the integer in flash.
Returns	--
Description	Writes a value to an integer that is stored in flash.
File	emeter-flash.c
Comments	--

*void flash_write_int8(int8_t *ptr, int8_t value)*	
Parameters	*ptr* - Address of the byte in flash that is to be rewritten. *value* - Value to be written to the byte in flash.
Returns	--
Description	Writes a value to a byte that is stored in flash.
File	emeter-flash.c
Comments	--

*void flash_write_int16(int16_t *ptr, int16_t value)*	
Parameters	*ptr* - Address of the integer in flash that is to be rewritten. *value* - Value to be written to the integer in flash.
Returns	--
Description	Writes a value to an integer that is stored in flash.
File	emeter-flash.c
Comments	--

*void flash_write_int32(int32_t *ptr, int32_t value)*	
Parameters	*ptr* - Address of the long in flash that is to be rewritten. *value* - Value to be written to the long in flash.
Returns	--
Description	Writes a value to a long that is stored in flash.
File	emeter-flash.c
Comments	--

*void flash_memcpy(char *ptr, char *from, int len)*	
Parameters	*ptr* - Address of the destination of the copied bytes. *from* - Source of the copied bytes. *len* - Total bytes to write.
Returns	--
Description	Copies bytes from one location to another.
File	emeter-flash.c
Comments	--

*void flash_replace16(int16_t *ptr, int16_t word)*	
Parameters	*ptr* - Address of integer to be replaced. *word* - Value to replace the desired integer in flash with.
Returns	--
Description	Makes the flash look like EEPROM. This function erases and replaces just one word. It erases SEGA and then images SEGB to SEGA. It then erases SEGB and copies from SEGA back to SEGB all 128 bytes except the one to be replaced.
File	emeter-flash.c
Comments	--

*void flash_replace32(int32_t *ptr, int32_t word)*	
Parameters	*ptr* - Address of long to be replaced. *word* - Value to replace the desired long in flash with.
Returns	--
Description	Makes the flash look like EEPROM. This function erases and replaces just one long word. It erases SEGA and then images SEGB to SEGA. It then erases SEGB and copies from SEGA back to SEGB all 128 bytes except the one to be replaced.
File	emeter-flash.c
Comments	--

void flash_secure(void)	
Parameters	--
Returns	--
Description	Locks the flash so that it is now only read-only.
File	emeter-flash.c
Comments	--

*void add_sumcheck(void *buf, int len)*	
Parameters	*buf* - Buffer to add a sumcheck to. *len*- The size of the message.
Returns	--
Description	Add a one byte sumcheck to the passed message, in the byte after the message.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

*int test_sumcheck(const void *buf, int len)*	
Parameters	*buf* - Buffer to check the sumcheck of. *len* - The size of the message.
Returns	1 if the message is valid. 0 if the message fails the test.
Description	check the passed message, which must include a one byte sumcheck, is OK and has not been corrupted.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void multirate_energy_pulse(void)	
Parameters	--
Returns	--
Description	Updates the log of the total number of energy pulses for the current tariff of the current bill-cutoff (bill cycle).
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

int find_next_cutoff_date(void)	
Parameters	--
Returns	The index into the array of cutoff-structures where the information about the next cutoff date is stored.
Description	Find the slot number in memory of cutoff dates for the next cutoff date (billing cycle) from today.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

int find_previous_cutoff_date(void)	
Parameters	--
Returns	The index into the array of cutoff-structures where the information about the previous cutoff date is stored.
Description	Find the slot number in the array of cutoff dates for the previous cutoff date (billing cycle) from today.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void new_tariff_day(void)	
Parameters	--
Returns	--
Description	When it is a new day, update the tariff to the new day's schedule, which is based on whether it is a holiday or what day of week it is, and clear the daily logged parameters. Also, check if the new day is on a new billing cycle compared to the previous day, and if it is, update the current billing cycle.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void new_tariff_minute(void)	
Parameters	--
Returns	--
Description	Check if the meter should be running on a different schedule after each minute. If it should be on a new schedule, update the tariff to the new schedule's tariff and store the total accumulated energy of the previous tariff to memory. Read the total accumulated energy of the new tariff and start accumulating energy from this starting point.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

int read_history_slot(int slot, int tariff)	
Parameters	*slot* - The index into the array of cutoff-structures that corresponds to the billing cycle whose energy accumulation is desired to be read. *tariff*- The tariff type whose energy accumulation is desired to be read.
Returns	0 if the read is successful. (-1) if the read is not successful.
Description	Reads the accumulated energy history for the particular billing cycle at the selected tariff and stores it in the current_history structure.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

int write_history_slot(int slot, int tariff)	
Parameters	*slot* - An index into the array of cutoff-structures that corresponds to the billing cycle whose energy accumulation value is desired to be updated. *tariff* - The tariff type whose energy accumulation is desired to be updated.
Returns	The output of running the iicEEPROM_write function.
Description	Writes the accumulated energy stored in the current_history structure to the energy accumulation data for the selected tariff at the selected billing cycle.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void tariff_management(void)	
Parameters	--
Returns	--
Description	If it is a new day or minute, update the tariff accordingly.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void tariff_initialise(void)	
Parameters	--
Returns	--
Description	Initialize tariff information after starting from reset.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

void multirate_align_with_rtc(void)	
Parameters	--
Returns	--
Description	Align the multi-rate activities with the new time and date after the RTC has just changed. The meter may have hopped between cutoff dates so a full re-alignment with the new date is necessary.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

*int multirate_put(uint8_t *msg)*	
Parameters	*msg* - Received message from GUI.
Returns	Returns 0 if the message received from the GUI requests to change a parameter not supported. Otherwise, return the output from the iicEEPROM_write function.
Description	Replace multirate parameters with parameters sent from the GUI.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

*int multirate_get(uint8_t *msg, uint8_t *txmsg)*	
Parameters	*msg* - Received partial message from GUI. *txmsg* - Message to be sent to GUI with the requested parameters.
Returns	8 if successfully read tariff, holiday, or cutoff parameters. 10 if successfully read information about the weekday schedule type. 4 otherwise.
Description	Send requested multirate parameters to the GUI.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

*int multirate_clear_usage(uint8_t *msg)*	
Parameters	*msg* - Received message from GUI.
Returns	0 if the message received from the GUI requests to clear a parameter not supported. Otherwise, return the output from the iicEEPROM_write function.
Description	Clear particular multirate **Parameters**
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

*int multirate_get_usage(uint8_t *msg, uint8_t *txmsg)*	
Parameters	*msg* - Received partial message from GUI. *txmsg* - Message to be sent to GUI with the requested usage.
Returns	10 if successfully read information about the daily peak for a particular day. 12 if successfully read information about the power consumption for a particular tariff at a particular billing cycle. 4 otherwise.
Description	Send requested multirate parameters to the GUI.
File	emeter-multirate.c
Comments	This function is available only if multirate support is enabled.

static __inline__ long int labs(long int __x)	
Parameters	*x* - Number to take the absolute value of.
Returns	The absolute value of x.
Description	Calculates the absolute value of a function.
File	emeter-main.c
Comments	--

int record_meter_failure(int type)	
Parameters	*type* - Parameter that should be a value between 0 and 15, specifying the unrecoverable error type to be recorded in the failures word in flash.
Returns	1 if the function completes successfully.
Description	Records a meter failure and the type of error.
File	emeter-main.c
Comments	Don't worry about the time taken to write to flash - we are recording a serious error condition. This function is available only if SELF_TEST_SUPPORT is defined.

int record_meter_warning(int type)	
Parameters	*type* - Should be a value between 0 and 15, specifying the warning type to be recorded in the recoverable failures word in flash.
Returns	1 if the function completes successfully.
Description	Records a meter warning and the type of warning.
File	emeter-main.c
Comments	Don't worry about the time taken to write to flash - we are recording a serious error condition. This function is available only if SELF_TEST_SUPPORT is defined.

void test_battery(void)	
Parameters	--
Returns	--
Description	Tests battery.
File	emeter-main.c
Comments	This function is available only if BATTERY_MONITOR_SUPPORT is defined.

void set_io_expander(int what, int which)	
Parameters	*what* - Parameter that determines how to change the expanded IO's state. If "what" is less than 0, the current value for the IO's state is combined by a logical AND with the bitwise-not of the parameter "which". If "what" is greater than 0, the current value for the IO's state is combined by a logical OR with the parameter "which". If "what" equals 0, the current value of the IO's state is set to the variable "which". *which* - Variable used to change the IO state.
Returns	--
Description	Supports the use of a device like the 74HC595 to expand the number of output bits available on the lower pin count MSP430s.
File	emeter-main.c
Comments	This function is available only if IO_EXPANDER_SUPPORT is defined.

static __inline__ int keypad_debounce(void)	
Parameters	--
Returns	1 if the foreground should be triggered. 0 if it should not be triggered.
Description	Debouncing for 1 to 4 keys.
File	emeter-main.c
Comments	This function is available only if BASIC_KEYPAD_SUPPORT and x__MSP430__ are defined or if CUSTOM_KEYPAD_SUPPORT is defined.

void main(void)	
Parameters	--
Returns	--
Description	The main function. When the background process has finished a block processing operation, the main function calls a function that uses the parameters calculated by the background process to calculate the metering parameters. It also calls a function to update the display and perform other housekeeping tasks.
File	emeter-main.c
Comments	--

int32_t current_consumed_active_energy(int ph)	
Parameters	*ph* - Which phase's active energy is to be returned.
Returns	Total consumed active energy of the selected phase.
Description	Returns the total consumed active energy of the selected phase.
File	emeter-main.c
Comments	--

power_t current_active_power(int ph)	
Parameters	*ph* - Which phase's active power is to be returned.
Returns	Active power of the selected phase.
Description	Returns the active power of the selected phase.
File	emeter-main.c
Comments	--

int32_t current_consumed_reactive_energy(int ph)	
Parameters	*ph* - Which phase's total reactive energy is to be returned.
Returns	Total consumed reactive energy of the selected phase.
Description	Returns the total consumed reactive energy of the selected phase.
File	emeter-main.c
Comments	This function is not available if REACTIVE_POWER_SUPPORT is not defined.

power_t current_reactive_power(int ph)	
Parameters	*ph* - Which phase's reactive power is to be returned.
Returns	Reactive power of the selected phase.
Description	Returns the reactive power of the selected phase.
File	emeter-main.c
Comments	This function is not available if REACTIVE_POWER_SUPPORT is not defined.

power_t current_apparent_power(int ph)	
Parameters	*ph* - Which phase's apparent power is to be returned.
Returns	Apparent power of the selected phase.
Description	Returns the apparent power of the selected phase.
File	emeter-main.c
Comments	This function is not available if APPARENT_POWER_SUPPORT is not defined.

power_t current_fundamental_active_power(int ph)	
Parameters	*ph* - Which phase's fundamental active power is to be returned.
Returns	Fundamental active power of the selected phase.
Description	Returns the fundamental active power of the selected phase.
File	emeter-main.c
Comments	This function is not available if FUNDAMENTAL_ACTIVE_POWER_SUPPORT is not defined.

power_t current_fundamental_reactive_power(int ph)	
Parameters	*ph* - Which phase's fundamental reactive power is to be returned.
Returns	Fundamental reactive power of the selected phase.
Description	Returns the fundamental reactive power of the selected phase.
File	emeter-main.c
Comments	This function is not available if FUNDAMENTAL_REACTIVE_POWER_SUPPORT is not defined.

int32_t current_power_factor(int ph)	
Parameters	*ph* - Which phase's power factor is to be returned.
Returns	Power factor of the selected phase.
Description	Returns the power factor of the selected phase.
File	emeter-main.c
Comments	This function is not available POWER_FACTOR_SUPPORT is not defined.

rms_voltage_t current_rms_voltage(int ph)	
Parameters	*ph* - Which phase's RMS voltage is to be returned.
Returns	Rms voltage of the selected phase.
Description	Returns the RMS voltage of the selected phase.
File	emeter-main.c
Comments	This function is not available if VRMS_SUPPORT is not defined.

rms_current_t current_rms_current(int ph)	
Parameters	*ph* - Which phase's RMS current is to be returned.
Returns	Rms current of the selected phase.
Description	Returns the RMS current of the selected phase.
File	emeter-main.c
Comments	This function is not available if IRMS_SUPPORT is not defined.

int32_t current_mains_frequency(int ph)	
Parameters	*ph* - Which phase's mains frequency reading is to be returned.
Returns	Mains frequency reading of the selected phase.
Description	Returns the mains frequency reading of the selected phase.
File	emeter-main.c
Comments	This function is not available if MAINS_FREQUENCY_SUPPORT is not defined.

void switch_to_normal_mode(void)	
Parameters	--
Returns	--
Description	Take the necessary actions when switching to normal mode.
File	emeter-main.c
Comments	--

void switch_to_limp_mode(void)	
Parameters	--
Returns	--
Description	Take the actions necessary when switching to limp-mode.
File	emeter-main.c
Comments	--

void switch_to_powerfail_mode(void)	
Parameters	--
Returns	--
Description	Take the necessary actions when switching to powerfail-mode.
File	emeter-main.c
Comments	--

void phase_active_energy_pulse_start(void)	
Parameters	--
Returns	--
Description	Sets the active energy output to logic high. This function with the phase_active_energy_pulse_end function creates a pulse.
File	emeter-main.c
Comments	This is the one-phase version needed to output pulses. This function is available only if PER_PHASE_ACTIVE_ENERGY_SUPPORT is defined.

void phase_active_energy_pulse_end(void)	
Parameters	--
Returns	--
Description	Sets the active energy output to logic low. This function with the phase_active_energy_pulse_start function creates a pulse.
File	emeter-main.c
Comments	This is the one-phase version needed to output pulses. This function is available only if PER_PHASE_ACTIVE_ENERGY_SUPPORT is defined.

void phase_active_energy_pulse_start(int ph)	
Parameters	*ph* - Which phase's active energy pulse to output a pulse on.
Returns	--
Description	Sets the active energy output for a particular phase to logic high. This function with the phase_active_energy_pulse_end function creates a pulse when the active energy for a phase reaches the user-defined active-energy pulse threshold.
File	emeter-main.c
Comments	This function is the multi-phase version needed to output active energy pulses. This function is available only if PER_PHASE_ACTIVE_ENERGY_SUPPORT is defined.

void phase_active_energy_pulse_end(int ph)	
Parameters	*ph* - Which phase's active energy pulse to output a pulse on.
Returns	--
Description	Sets the active energy output for a particular phase to logic low. This function with the phase_active_energy_pulse_start creates a pulse when the active energy for a phase reaches the user-defined active-energy pulse threshold.
File	emeter-main.c
Comments	This is the multi-phase version needed to output active energy pulses. This function is available only if PER_PHASE_ACTIVE_ENERGY_SUPPORT is defined.

void phase_reactive_energy_pulse_start(void)	
Parameters	--
Returns	--
Description	Sets the reactive energy output to logic high. This function with the phase_reactive_energy_pulse_end function creates a pulse.
File	emeter-main.c
Comments	This function is available only if PER_PHASE_REACTIVE_ENERGY_SUPPORT is defined.

void phase_reactive_energy_pulse_end(void)	
Parameters	--
Returns	--
Description	Sets the reactive energy output to logic low. This function with the phase_reactive_energy_pulse_start function creates a pulse.
File	emeter-main.c
Comments	This function is available only if PER_PHASE_REACTIVE_ENERGY_SUPPORT is defined.

void total_active_energy_pulse_start(void)	
Parameters	--
Returns	--
Description	Sets the total active energy output to logic high. This function with the total_active_energy_pulse_end function creates a pulse.
File	emeter-main.c
Comments	This function is available only if TOTAL_ACTIVE_ENERGY_SUPPORT is defined.

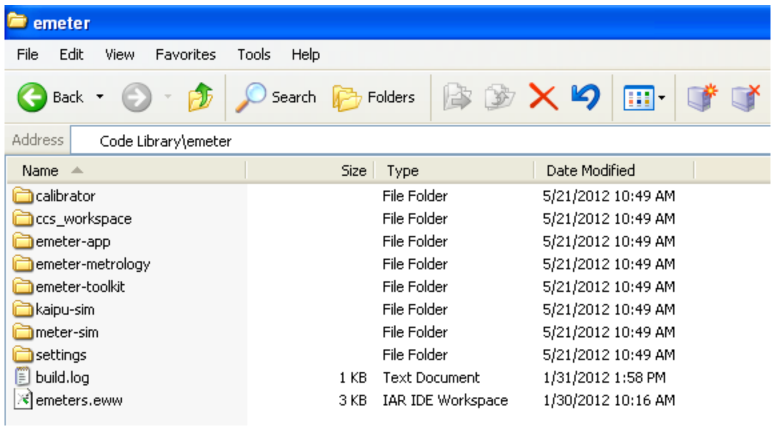
void total_active_energy_pulse_end(void)	
Parameters	--
Returns	--
Description	Sets the total active energy output to logic low. This function with the phase_active_energy_pulse_start function creates a pulse.
File	emeter-main.c
Comments	This function is available only if TOTAL_ACTIVE_ENERGY_SUPPORT is defined.

void total_reactive_energy_pulse_start(void)	
Parameters	--
Returns	--
Description	Sets the total reactive energy output to logic high. This function with the total_reactive_energy_pulse_end function creates a pulse.
File	emeter-main.c
Comments	This function is available only if TOTAL_REACTIVE_ENERGY_SUPPORT is defined.

void total_reactive_energy_pulse_end(void)	
Parameters	--
Returns	--
Description	Sets the total reactive energy output to logic low. This function with the phase_reactive_energy_pulse_start function creates a pulse.
File	emeter-main.c
Comments	This function is available only if TOTAL_REACTIVE_ENERGY_SUPPORT is defined.

3 Loading the Example Code

The source code is developed in the IAR environment using IAR compiler version 6.x. The project files cannot be opened in earlier versions of IAR. If a version later than 6.x versions is used, a prompt to create a back-up is displayed when the project is loaded, and the user can click YES to proceed. To run the code, first navigate to the Code Library\emeter directory, which is shown in Figure 1.


emeter
File Edit View Favorites Tools Help
Back Search Folders
Address Code Library\emeter

Name	Size	Type	Date Modified
calibrator		File Folder	5/21/2012 10:49 AM
ccs_workspace		File Folder	5/21/2012 10:49 AM
emeter-app		File Folder	5/21/2012 10:49 AM
emeter-metrology		File Folder	5/21/2012 10:49 AM
emeter-toolkit		File Folder	5/21/2012 10:49 AM
kaipu-sim		File Folder	5/21/2012 10:49 AM
meter-sim		File Folder	5/21/2012 10:49 AM
settings		File Folder	5/21/2012 10:49 AM
build.log	1 KB	Text Document	1/31/2012 1:58 PM
emeters.eww	3 KB	IAR IDE Workspace	1/30/2012 10:16 AM

Figure 1. Loading the Example Code

The folders emeter-app, emeter-metrology, and emeter-toolkit each contain multiple project files. Within these folders, the projects emeter-app_xxxx, emeter-metrology_xxxx, and emeter-toolkit_xxxx should always be chosen, where xxxx represents the device family of the desired meter EVM. For first time use, it is recommended that all three projects be completely rebuild. To do this, first Open the emeters.eww workspace, select the emeter-toolkit-xxxx.ewp project from the project tabs beneath the current project's files, and do a rebuild all. Then select the emeter-metrology_xxxx .ewp project from the project tabs beneath the current project's files and do a rebuild all. Finally, select the emeter-app_xxxx.ewp project from the project tabs beneath the current project's files, choose rebuild all, and load this on to the desired meter. For the compilation to successfully complete, the projects should always be rebuilt in this order. In the snapshots below, the rebuilding process is shown for the msp430afe253 meter; that is, xxxx=afe253.

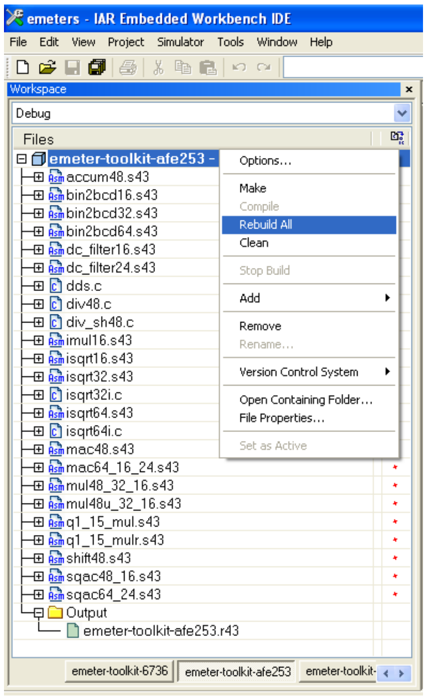


Figure 2. Toolkit Compilation in IAR

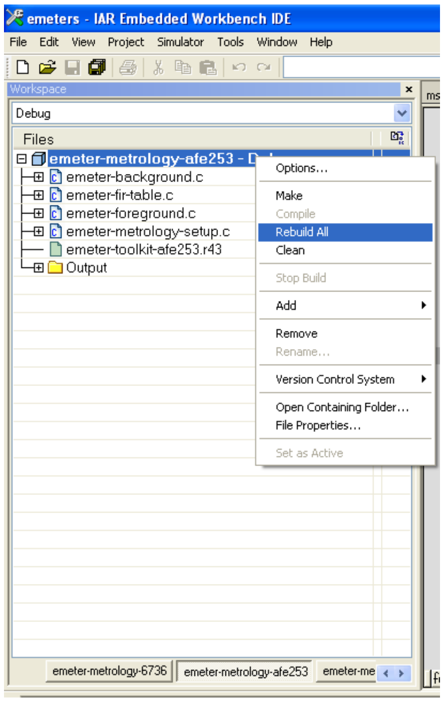


Figure 3. Metrology Compilation in IAR

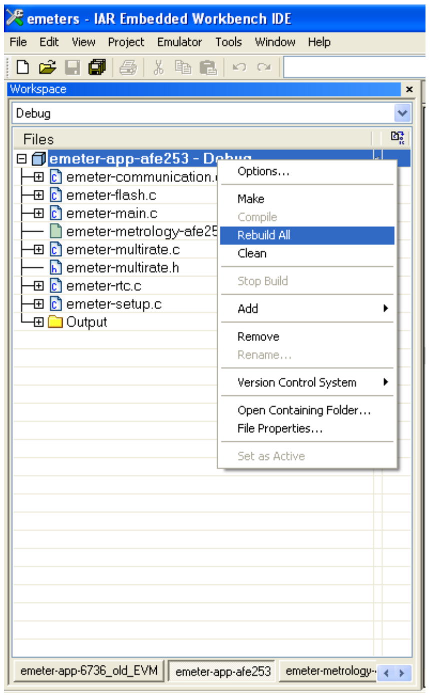


Figure 4. Application Compilation in IAR

After the main project has been rebuilt, load it on to the EVM by clicking Download and Debug and then pressing Go from the Debug menu.

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 Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
		TI E2E Community Home Page	e2e.ti.com