Source Code

1. *es_version.h*

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_version.h
// Version control, Tools info and generic info
//
// 1/5/16 - Version 1.0 - In progress
// Tools list:
// 1. Compiler: TI v5.2.7
// 2. IDE: Code Composer Studio v6.1.2
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_COMMON_ES_VERSION_H_
#define SRC_COMMON_ES_VERSION_H_

char VERSION[]  = "CAPE, Version 1.00";

/***** Important notes  ***********************************************
 * 1. FILE NAMING PROTOCOL
 * All files written by us have a prefix of es (for eg: es_cape.c etc..)
 *
 * 2. LIBRARIES USED:
 *
 * I. From TivaWare (TI): http://www.ti.com/tool/sw-tm4c
 *  1. DriverLib (Device Drivers for TIVA)
 *  2. IQMathLib (Floating point math using fixed point notation)
 *  3. Graphics Library (Graphical library for display)
 *
 * II. ARM CMSIS DSP library
 * http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-
standard.php
 *
 * III. LwIP ethernet stack
 * savannah.nongnu.org/projects/lwip/
 *
 *
 */



#endif /* SRC_COMMON_ES_VERSION_H_ */
```

2. es_cape.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_cape.c Main project file
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/******************* ABOUT THIS PROJECT **********************************
 * This project was created as a part of the final project for ESD Spring 2016
 * by Meher Jain and Vivek Sankaranarayanan.
 *
 * The major fucntionality of the project:
 *  1. The project is called CAPE (Compact Analyser for Power Equipment). This
 *     basically measures the AC voltage and current and measures the harmonics
 *     in the power drawn by the load.
 *  2. It measures a bunch of AC metrics in addition to voltage and current
 *     harmonics. (Vrms, Irms, Vpeak, Ipeak etc... See measurement.c for
 *     complete list)
 *  3. The project uses a KENTEC TFT touchscreen (resistive) LCD that displays
 *     AC_metrics, Voltage/Current Fourier spectrum, Waveforms. See gui.c
 *     for more information
 *  4. The project also has an lwIP (ethernet) based datalogger. See
 *     enet_datalogger.c
 *
 *     For libraries, tool info and file naming conventions see version.h
 */

/******************* Header files  ***************************************/
/* device support */
#include <es_enet_datalogger.h>
#include <es_fft_compute.h>
#include <es_gui.h>
#include <es_measurements.h>
#include <es_pll.h>
#include "device.h"
#include "drivers/pinout.h"

/* ac measurements */
#include "drivers/touch.h"
#include "utils/uartstdio.h"
/*************************************************************************/

/***************** Global variables ************************************/
volatile uint8_t scheduler_flag; /* Flag to schedule events */
uint32_t sys_clk; /* system clock counts */
/*************************************************************************/

/* desc: Initialises all peripherals used in the project
 * args: none
 * ret : none
 */
static void peripheral_init()
{
  /* Enable the GPIO port that is used for the on-board LED. */
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
```

```
/* Set the direction as output, and enable the GPIO pin for IO function */
ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0);
ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);

/* Configure the device pins.*/
PinoutSet(true, false);

/* Configure UART.*/
UARTStdioConfig(0, 115200, sys_clk);

/* Uart test */
UARTprintf("Uart functional...\n\r");

/*
 * Systick Initialisation
 */
ROM_SysTickPeriodSet(SYSTICK_COUNT_FOR_NOMINAL_LINE_FREQUENCY); // 3906
ROM_SysTickIntEnable();
ROM_SysTickEnable();

/*
 * PWM initialisation
 * PWM0->PF1: 20KHz
 */
/* Pin */
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
/* Select PWM functionality for the pins   */
ROM_GPIOPinConfigure(GPIO_PF1_M0PWM1);
/* Configure the PWM functionality for the pin */
ROM_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);

/* Peripheral */
/* PWM0 in count down mode */
ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
/*
 Set the PWM period to 20000Hz.  To calculate the appropriate parameter
 use the following equation: N = (1 / f) * SysClk.  Where N is the
 function parameter, f is the desired frequency, and SysClk is the
 system clock frequency.
 In this case you get: (1 / 20kHz) * 120MHz = 6000 cycles.  Note that
 the maximum period you can set is 2^16.
 */
ROM_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 6000);
/* zero duty initialy */
ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, 0);
/* Enable the PWM0 output signal (PF1).*/
ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
/* Enables the PWM generator block.*/
ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);

/*
 * General purpose timer 0: Capture mode
 */
/* T0CCP0 Initialisation (PL4-65 breadboard) */
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
ROM_GPIOPinConfigure(GPIO_PL4_T0CCP0);
ROM_GPIOPinTypeTimer(GPIO_PORTL_BASE, GPIO_PIN_4);

/* Timer 0 Capture Initialisation */
```

```
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  TIMER0_CC_R = 0x00; /* Clocked with SYSCLK (120MHz) */
  TIMER0_CTL_R = 0; /* Timer disabled */
  TIMER0_CFG_R = TIMER_CFG_16_BIT; /* 16 bit configuration */
//  TIMER0_CFG_R = TIMER_CFG_32_BIT_TIMER; /* 32 bit configuration */
  /* Up count, Capture mode, Edge Time, Capture mode, Capture mode interrupt enable */
  TIMER0_TAMR_R =
      TIMER_TAMR_TACDIR + TIMER_TAMR_TACMR + TIMER_TAMR_TAMR_CAP/* + TIMER_TAMR_TAPWMIE*/;
  TIMER0_CTL_R = TIMER_CTL_TAEVENT_NEG; /* Capture on falling edge */
  TIMER0_IMR_R = TIMER_IMR_CAEIM; /* Capture mode event interrupt enable */
  TIMER0_ICR_R = 0xFFFFFFFF; /* Clear interrupt status */
  TIMER0_TAILR_R = 0xFFFF; /* 2^16 is the upper bound */
  TIMER0_TAPR_R = 0xFF; /* 2^8 is upper bound */
  TIMER0_CTL_R |= TIMER_CTL_TAEN; /* Enable Timer */
}

/* Application main file */
int main(void)
{
  /* For ethernet. Checking clock
   Make sure the main oscillator is enabled because this is required by
   the PHY.  The system must have a 25MHz crystal attached to the OSC
   pins. The SYSCTL_MOSC_HIGHFREQ parameter is used when the crystal
   frequency is 10MHz or higher.
   */
  SysCtlMOSCConfigSet(SYSCTL_MOSC_HIGHFREQ);

  /* Set the clocking to run directly from the crystal at 120MHz. */
  sys_clk = ROM_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
  SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
  SYSCTL_CFG_VCO_480), 120000000);

  /* enables the floating point engine for the processor */
  FPUEnable();
  /* FPU lazy stacking enables optimization when interrupt is called */
  FPULazyStackingEnable();

//  /* FFT test */
//  generate_input();

  /* Initialise all global stuff */
  peripheral_init();
  pll_init(&pll_s);
  scheduler_flag = false;
  init_adc();
  gui_init();
  enet_init();

  /* Enable Interrupts at NVIC level */
  ROM_IntEnable(INT_TIMER0A);
  /* enable interrupt processing at CPU level */
  ROM_IntMasterEnable();

  /* connect to server */
  enet_server_connect();

  fft_VI_samples_collect = 1;

  /* Enable initial trigger for touchscreen ADC */
  ROM_ADCProcessorTrigger(ADC0_BASE, 3);
```

```c
/* Begin endless loop */
while (1)
{
  /* Check if ADC0 conversion completed */
  if (ROM_ADCIntStatus(ADC0_BASE, 3, false))
  {
    ROM_ADCIntClear(ADC0_BASE, 3);
    /* Retrigger for next time */
    ROM_ADCProcessorTrigger(ADC0_BASE, 3);
    /* Call touch screen handler to process ADC results */
    TouchScreenHandler();
  }

  /* GrLib function that checks if any widgets have been triggered */
  WidgetMessageQueueProcess();

  /* Execute scheduled events */
  if (scheduler_flag == true)
  {
    /* reset scheduler flag in beginning itself before any event executed
     * so that any stalled event may cause future events to be lost
     */
    scheduler_flag = false;

    /* PLL EVENT */
    phase_locked_loop(&pll_s);

    /* FFT EVENT */
    if (fft_counter == FFT_EVENT_COUNTER_ELAPSED)
    {
      fft_counter = 0;
      /* disable sample collection when fft being computed for the existing samples*/
      fft_VI_samples_collect = false;
      fft_compute();
      /* renable sample collection */
      fft_VI_samples_collect = true;
    }
    else
    {
      fft_counter++;
    }

    /* DISPLAY EVENT */
    if (counter_display_event == DISPLAY_EVENT_COUNTER_ELAPSED)
    {
      counter_display_event = 0;
      gui_update();

      if (g_iPage == PAGE_METRICS)
      {
        // Updating Vrms //
        sprintf(display_update_buffer1, "%.1f V",
            ac_metrics.Vac.norm_rms * V_FULL_SCALE / IQ24toFloat);
        CanvasTextSet(&g_sVrmsValue, (const char * )display_update_buffer1);

        // Updating Irms //
        sprintf(display_update_buffer2, "%.1f A",
            ac_metrics.Iac.norm_rms * I_FULL_SCALE / IQ24toFloat);
        CanvasTextSet(&g_sIrmsValue, (const char * )display_update_buffer2);

        // Updating Frequency //
```

```
    sprintf(display_update_buffer3, "%.1f Hz",
        120000000.0 / pll_s.freq_in_cap_counts);
    CanvasTextSet(&g_sFreqValue, (const char * )display_update_buffer3);

    // Updating Power Factor //
    sprintf(display_update_buffer4, "%.2f",
        ac_metrics.P_PowerFactor / IQ24toFloat);
    CanvasTextSet(&g_sPFValue, (const char * )display_update_buffer4);

    // Updating Apparent Power //
    sprintf(display_update_buffer5, "%.1f",
        ac_metrics.P_apparent * P_FULL_SCALE / IQ24toFloat);
    CanvasTextSet(&g_sP_apparent_val,
        (const char * )display_update_buffer5);

    // Updating active Power //
    sprintf(display_update_buffer6, "%.1f",
        ac_metrics.P_active * P_FULL_SCALE / IQ24toFloat);
    CanvasTextSet(&g_sP_active_val,
        (const char * )display_update_buffer6);

    // Updating THD Voltage //
    sprintf(display_update_buffer7, "%.1f %%", ac_metrics.Vthd);
    CanvasTextSet(&g_sTHDv_val, (const char * )display_update_buffer7);

    // Updating THD Current //
    sprintf(display_update_buffer8, "%.1f %%", ac_metrics.Ithd);
    CanvasTextSet(&g_sTHDi_val, (const char * )display_update_buffer8);

    // Updating Phase  //
    sprintf(display_update_buffer9, "%.1f dg",
        ac_metrics.Phase_shift * (180 / PI) / IQ24toFloat);
    CanvasTextSet(&g_sPhase_val, (const char * )display_update_buffer9);

    // Vpeak
    sprintf(display_update_buffer10, "%.1f V",
        ac_metrics.V_Peak * V_FULL_SCALE / IQ24toFloat);
    CanvasTextSet(&g_sVpeak_val, (const char * )display_update_buffer10);
    ac_metrics.V_Peak = 0;

    // Ipeak
    sprintf(display_update_buffer11, "%.1f A",
        ac_metrics.I_Peak * I_FULL_SCALE / IQ24toFloat);
    CanvasTextSet(&g_sIpeak_val, (const char * )display_update_buffer11);
    ac_metrics.I_Peak = 0;

    WidgetPaint(WIDGET_ROOT);  // painting the updated canvases
  }
  else if (g_iPage == PAGE_VOLTAGE_SPECTRUM)
  {
    Display_FreqSpectrum(VOLTAGE_SPECTRUM);
  }
  else if (g_iPage == PAGE_CURRENT_SPECTRUM)
  {
    Display_FreqSpectrum(CURRENT_SPECTRUM);
  }
  else if (g_iPage == PAGE_TIME_DOMAIN_SIGNAL)
  {
    Display_TimeDomain();
  }
}
```

```
      else
      {
        counter_display_event++;
      }

      /* ENET event */
      if (nw_update_timer == ENET_EVENT_COUNTER_ELAPSED)
      {
        nw_update_timer = 0;
        enet_metrics_log();
      }
      else
        nw_update_timer++;
    }
  } /* end of while(1) */
} /* end of main */

/*
 * desc: systick interrupt handler
 *       This interrupt occurs at (line_frequency * 512 Hz)
 *       The occurrence for this interrupt is not fixed and it can vary from
 *        cycle to cycle if line frequency varies. The systick interrupt
 *        interval is modified by pll algorithm running every line cycle
 *       Major tasks in the interrupt:
 *       1. sine_index control and sine wave generation
 *       2. AC_metrics computation (Vrms, Irms, Power)
 *       3. FFT sample collection
 *       4. Waveform sample collection
 *       5. Update scheduler flag
 *       6. Upadte ethernet timers
 * args: none
 * ret : none
 */
void sys_tick_handler()
{
  volatile int32_t radians;
  volatile int32_t sin;
  volatile uint16_t pwm_counts;

//  HWREG(GPIO_PORTN_BASE + (1 << 2)) = 1;

  /* Calculate radians */
  /* radians  = (sine_index * 2 * pi) / 512 */
  radians = _IQmpy(_IQ(PI)<<1,
      _IQ21toIQ(_IQ21div(_IQ21(pll_s.sine_index),_IQ21(SINE_SAMPLE_SIZE))));
  pll_s.sine_index = (pll_s.sine_index + 1) & (SINE_SAMPLE_SIZE - 1);

  /* trigger adc on even sine index */
  if ((pll_s.sine_index & 0x01) == 0)
  {
    ROM_ADCProcessorTrigger(ADC1_BASE, 0);
  }
  else
  {
    /* read result on odd sine index */
    ROM_ADCSequenceDataGet(ADC1_BASE, 0, &ac_raw_adc_counts[0]);
    ROM_ADCIntClear(ADC1_BASE, 0);

    /* Collect VI samples for FFT if collect flag set */
    if (fft_VI_samples_collect == true)
    {
```

```
      /* Build the fft samples array for current and voltage */
      norm_Vinst_IQ_samples[pll_s.sine_index / 2] =
      _Q12toIQ24((int32_t)ac_raw_adc_counts[0]) - _IQ(ADC_LEVEL_SHIFT);
      norm_Iinst_IQ_samples[pll_s.sine_index / 2] =
      _Q12toIQ24((int32_t)ac_raw_adc_counts[1]) - _IQ(ADC_LEVEL_SHIFT);
    }

    /* collect samples for waveform. 128 samples per cycle for the display
     * Skip alternate samples
     */
    if (skip_sample == 0)
    {
      V_waveform_buffer[wave_index] =
      _Q12toIQ24((int32_t)ac_raw_adc_counts[0]) - _IQ(ADC_LEVEL_SHIFT);
      I_waveform_buffer[wave_index] =
      _Q12toIQ24((int32_t)ac_raw_adc_counts[1]) - _IQ(ADC_LEVEL_SHIFT);
      wave_index++;
      /* wrap around the collection buffers */
      if (wave_index == DISPLAY_SAMPLE_SIZE)
        wave_index = 0;
    }
    skip_sample ^= 1;

    /* Square and accumulate adc channels (after removing offset) */
    ac_metrics.Vac.norm_acc += _IQmpy(
        (_Q12toIQ24(((int32_t)ac_raw_adc_counts[0]))-_IQ(ADC_LEVEL_SHIFT)),
        (_Q12toIQ24((int32_t)ac_raw_adc_counts[0])-_IQ(ADC_LEVEL_SHIFT)));

    ac_metrics.Iac.norm_acc += _IQmpy(
        (_Q12toIQ24(((int32_t)ac_raw_adc_counts[1]))-_IQ(ADC_LEVEL_SHIFT)),
        (_Q12toIQ24((int32_t)ac_raw_adc_counts[1])-_IQ(ADC_LEVEL_SHIFT)));

    /* power calculation */
    /* accumulate instantaneous power  */
    ac_metrics.P_inst_acc += _IQmpy(
        (_Q12toIQ24(((int32_t)ac_raw_adc_counts[0]))-_IQ(ADC_LEVEL_SHIFT)),
        (_Q12toIQ24((int32_t)ac_raw_adc_counts[1])-_IQ(ADC_LEVEL_SHIFT)));

    /* Calculating Vpeak and IPeak */
    if (_IQabs(_Q12toIQ24((int32_t)ac_raw_adc_counts[0])-_IQ(ADC_LEVEL_SHIFT))
        > ac_metrics.V_Peak)
      ac_metrics.V_Peak = _IQabs(
          _Q12toIQ24((int32_t)ac_raw_adc_counts[0])-_IQ(ADC_LEVEL_SHIFT));

    if (_IQabs(_Q12toIQ24((int32_t)ac_raw_adc_counts[1])-_IQ(ADC_LEVEL_SHIFT))
        > ac_metrics.I_Peak)
      ac_metrics.I_Peak = _IQabs(
          _Q12toIQ24((int32_t)ac_raw_adc_counts[1])-_IQ(ADC_LEVEL_SHIFT));
  }

//  /* For PLL debug */
//  if (pll_s.sine_index == 0)
//  {
//    /* toggle port here */
//    HWREG(GPIO_PORTN_BASE + (1 << 2)) ^= 1;
//  }

  /* Compute Rms parameters once every cycle */
  if (pll_s.sine_index == SINE_SAMPLE_SIZE - 1)
  {
    /* voltage rms: square root of mean of accumulated value */
```

```
    ac_metrics.Vac.norm_rms = _IQsqrt(ac_metrics.Vac.norm_acc >> 8);
    ac_metrics.Vac.norm_acc = 0;

    /* current rms: square root of mean of accumulated value */
    ac_metrics.Iac.norm_rms = _IQsqrt(ac_metrics.Iac.norm_acc >> 8);
    ac_metrics.Iac.norm_acc = 0;

    /* Active power: mean of accumulated value */
    ac_metrics.P_active = ac_metrics.P_inst_acc >> 8;
    ac_metrics.P_inst_acc = 0;

    /* Apparent power: Vrms * Irms */
    ac_metrics.P_apparent = _IQmpy(ac_metrics.Vac.norm_rms,
        ac_metrics.Iac.norm_rms);

    /* Calculate reactive power: sqrt(apparent^2 - active^2) */
    ac_metrics.P_reactive =
        _IQsqrt(
            _IQmpy(ac_metrics.P_apparent,ac_metrics.P_apparent) -
_IQmpy(ac_metrics.P_active,ac_metrics.P_active));

    /* Calculate Power Factor: active/apparent */
    ac_metrics.P_PowerFactor = _IQdiv(ac_metrics.P_active,
        ac_metrics.P_apparent);

    /* Calculate Phase Shift b/w sinusoid current and voltage waveform */
    ac_metrics.Phase_shift = _IQacos(
        _IQdiv(ac_metrics.P_active,ac_metrics.P_apparent));

    /* Scheduler Flag is set to true once a cycle  */
    scheduler_flag = true;

    /* Ethernet related activity required by lwip */
    lwIPTimer(SYSTICKMS);
  }

  /* Calculate sin */
  sin = _IQsin(radians);
  /* Level shift sin */
  sin += _IQ(1);
  /* Sin is now from 0 to 2. Scale to 0 to 1 */
  sin >>= 1;

  /* Sin is scaled from 0 to 1 such that:
   * sin (0) = 0.5
   * sin (pi/2) = 1
   * sine (pi) = 0.5
   */
  /* drive duty */
  pwm_counts = _IQmpy(sin, 6000);
  /* 0 duty cycle is not possible with the chip. Therefore,
   * we disable pulses at 0 duty cycle */
   */
  if (pwm_counts == 0)
    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, false);
  else
  {
    /* renable pulses for non zero duty cyles */
    ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, pwm_counts);
    ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
  }
```

```
//  HWREG(GPIO_PORTN_BASE + (1 << 2)) = 0;
}

/*
 * desc: timer interrupt handler
 *       This interrupt occurs at falling edge of the capture pin
 *       The zero crossing detector circuit is the source for the
 *        interrupt. Toggles pin whenever line voltage changes polarity
 *       Major tasks in the interrupt:
 *       1. Get frequency of the last line cycle in capture counts
 *       2. Phase_shift of generated_sine_wave with grid voltage
 * args: none
 * ret : none
 */
void timer0_isr_handler()
{
  /* Indication to pll algorithm that capture event occured */
  pll_s.capture_detected = true;
  /* raw counts. 24bit timer register */
  pll_s.cap_counts_now = (TIMER0_TAR_R & 0xFFFFFF);
  pll_s.freq_in_cap_counts = (pll_s.cap_counts_now - pll_s.cap_counts_prev)
      & 0xFFFFFF;
  pll_s.cap_counts_prev = pll_s.cap_counts_now;

  /* Phase difference is the value of sine_index at zero crossing interrupt */
  pll_s.phase_shift_index = pll_s.sine_index;

  TIMER0_ICR_R = 4;
}
```

3. *es_fft_compute.h*

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_fft_compute.h
// Header file for es_fft_compute.c
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_FFT_ES_FFT_COMPUTE_H_
#define SRC_FFT_ES_FFT_COMPUTE_H_

/******************** Header files  ***************************************/
#include "device.h"
#include "arm_math.h"
/*****************************************************************************/

/******************** Global Variables
*********************************************************/
extern uint8_t fft_VI_samples_collect;
extern uint8_t fft_counter;
extern _iq norm_Iinst_IQ_samples[FFT_LENGTH];
extern _iq norm_Vinst_IQ_samples[FFT_LENGTH];
extern float32_t fft_output_array_voltage[FFT_LENGTH];
extern float32_t fft_output_array_current[FFT_LENGTH];
extern float32_t sine_test_input[FFT_LENGTH];

/******************** constants
*************************************************************/
#define FFT_LENGTH                    256
#define FFT_EVENT_COUNTER_ELAPSED     5

/******************** Global functions
*********************************************************/
extern void fft_compute();
extern void generate_input();


#endif /* SRC_FFT_ES_FFT_COMPUTE_H_ */
```

4. es_fft_compute.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_fft_compute.c
// Top level fft file that calls CMSIS library modules
//
// Referenced code: CMSIS library is used for FFT computations
// Link: http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-
interface-standard.php
// Doc : http://www.keil.com/pack/doc/CMSIS/General/html/index.html
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/******************* Header files  ***************************************/
#include <es_fft_compute.h>
#include <es_measurements.h>
#include <math.h>
#include "arm_const_structs.h"
/*************************************************************************/

/****************** Global variables ***************************************/
//float32_t sine_test_input[FFT_LENGTH];
uint8_t fft_VI_samples_collect = 1;              /* flag to turn ON/OFF fft sample collection */
uint8_t fft_counter = 0;                         /* fft event counter */
_iq norm_Iinst_IQ_samples[FFT_LENGTH];           /* IQ instantaneous voltage samples */
_iq norm_Vinst_IQ_samples[FFT_LENGTH];           /* IQ instantaneous current samples */
float32_t fft_output_array_voltage[FFT_LENGTH];  /* fft ouptut array for voltage */
float32_t fft_output_array_current[FFT_LENGTH];  /* fft ouptut array for current */
float32_t fft_input_array[FFT_LENGTH];           /* float input array for fft algorithm */

arm_rfft_fast_instance_f32 rfft_fast_len256;     /* real fft object required tby the CMSIS
library */
/*************************************************************************/

/* desc: Call the underlying FFT drivers from CMSIS library
 * args: pointer to rfft object, pointer to input_array, pointer to output buffer
 * ret : none
 */
void fft(arm_rfft_fast_instance_f32 *S, float32_t *input_arr,
    float32_t *output_arr)
{

  /* Process the data through the CFFT/CIFFT module */
  arm_rfft_fast_f32(S, input_arr, output_arr, 0);

  /* Process the data through the Complex Magnitude Module for
   calculating the magnitude at each bin */
  arm_cmplx_mag_f32(output_arr, output_arr, FFT_LENGTH / 2);
}

/* desc: top level function that calls fft and computes THD
 * args: none
 * ret : none
 */
```

```
void fft_compute()
{
  int16_t i;
  float32_t sum_of_squares;

  /* initialisation */
  arm_rfft_fast_init_f32(&rfft_fast_len256, FFT_LENGTH);

  /* voltage array */
  for (i = 0; i < FFT_LENGTH; i++)
    fft_input_array[i] = norm_Vinst_IQ_samples[i] / 16777216.0;
//    fft_input_array[i] = sine_test_input[i];
  fft(&rfft_fast_len256, fft_input_array, fft_output_array_voltage);

  /* current array */
  for (i = 0; i < FFT_LENGTH; i++)
    fft_input_array[i] = norm_Iinst_IQ_samples[i] / 16777216.0;
  fft(&rfft_fast_len256, fft_input_array, fft_output_array_current);

  /* THD computation */
  /* THD = sqrt(V2^2 + V3^2 + V4^2 + V5^2 + ...) / V1
   * V1: Rms of fundamental component
   * V2, V3, V4.. : rms of harmonic components
   *
   * Note: Actual amplitudes are used instead of rms values. Needs to be
   * verified if this approach is correct
   */

  /* Voltage THD computation */
  /* the FFT results are scaled by FFT_LENGTH/2. Normalise them first */
  for (i = 0; i < FFT_LENGTH / 2; i++)
  {
    fft_output_array_voltage[i] = fft_output_array_voltage[i]
        / (FFT_LENGTH / 2);
  }

  /* now ingore first 2 points corresponding to dc and fundamental and do a sum of squares of the
rest */
  sum_of_squares = 0;
  for (i = 2; i < FFT_LENGTH / 2; i++)
  {
    sum_of_squares +=
        (fft_output_array_voltage[i] * fft_output_array_voltage[i]);
  }

  /* now, thd = sqrt(sum_of_squares)/v1 */
  ac_metrics.Vthd = sqrt(sum_of_squares) / fft_output_array_voltage[1];

  /* Current THD computation */
  /* the FFT results are scaled by FFT_LENGTH/2. Normalise them first */
  for (i = 0; i < FFT_LENGTH / 2; i++)
  {
    fft_output_array_current[i] = fft_output_array_current[i]
        / (FFT_LENGTH / 2);
  }

  /* now ingore first 2 points corresponding to dc and fundamental and do a sum of squares of the
rest */
  sum_of_squares = 0;
  for (i = 2; i < FFT_LENGTH / 2; i++)
  {
```

```
    sum_of_squares +=
        (fft_output_array_current[i] * fft_output_array_current[i]);
  }

  /* now, thd = sqrt(sum_of_squares)/v1 */
  ac_metrics.Ithd = sqrt(sum_of_squares) / fft_output_array_current[1];
}

//void generate_input()
//{
//  uint16_t i;
//
//  /* generate radians */
//  for (i = 0; i < FFT_LENGTH; i++)
//    sine_test_input[i] = i * 2 * PI / FFT_LENGTH;
//  /* generate sine */
//  for (i = 0; i < FFT_LENGTH; i++)
//    sine_test_input[i] = (sin(sine_test_input[i])
//        + 0.5 * sin(3 * sine_test_input[i]) + 0.25 * sin(5 * sine_test_input[i])
//        + 0.15 * sin(7 * sine_test_input[i]));
//}

//void fft_test()
//{
//  //arm_status status;
//  //float32_t maxValue;
//  uint16_t i = 0;
//
//  for(i = 0; i < FFT_LENGTH; i++)
//  {
//    //testOutput[i] = 0;
//  }
//  //status = ARM_MATH_SUCCESS;
////  /* Process the data through the CFFT/CIFFT module */
////  arm_rfft_fast_f32(&rfft_fast_len256, sine_test_input, testOutput, 0);
////  /* Process the data through the Complex Magnitude Module for
////  calculating the magnitude at each bin */
////  arm_cmplx_mag_f32(testOutput, testOutput, FFT_LENGTH/2);
//}
```

5. es_pll.h

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_pll.h
// Header file for es_pll.c//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_LINE_MON_ES_PLL_H_
#define SRC_LINE_MON_ES_PLL_H_

/******************** Header files  ***************************************/
#include "device.h"
/*****************************************************************************/

/******************** Structure prototypes
*****************************************************/
typedef struct
{
    uint32_t cap_counts_prev;
    uint32_t cap_counts_now;
    uint32_t freq_in_cap_counts;
    uint16_t phase_shift_index;
    uint16_t sine_index;
    uint8_t capture_detected;
}pll_t;

/******************** Global Variables
*********************************************************/
extern pll_t pll_s;

/** Constants
*******************************************************************************/
#define MAX_INCREMENTAL_CHANGE   50
#define SINE_SAMPLE_SIZE         512

/** function macros
*******************************************************************************/
/*
 * PHASE CHECK
 * 1. Phase lead: Inverter in positive half cycle at capture interrupt
 * 2. Phase lag: Inverter in negative half cycle at capture interrupt
 */
#define PHASE_LEAD(A)           ((A>0) && (A<=SINE_SAMPLE_SIZE/2))
#define PHASE_LAG(A)            ((A>SINE_SAMPLE_SIZE/2) && (A < (SINE_SAMPLE_SIZE)))

/******************** Function prototypes
*********************************************************/
extern void pll_init(pll_t *pll_s);
extern void phase_locked_loop(pll_t *pll);
#endif /* SRC_LINE_MON_ES_PLL_H_ */
```

6. es_pll.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_pll.c
// PLL algorithm
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/******************* Header files  ***************************************/
#include <es_pll.h>
/*****************************************************************************/

/***************** Global variables ****************************************/
pll_t pll_s;
/*****************************************************************************/

/* desc: Initialize ADC_1 for voltage/current samples
 * args: pointer to PLL structure
 * ret : none
 */
void pll_init(pll_t *pll_s)
{
  pll_s->cap_counts_now = 0;
  pll_s->cap_counts_prev = 0;
  pll_s->capture_detected = 0;
  pll_s->freq_in_cap_counts = 0;
  pll_s->phase_shift_index = 0;
}

/**
 * @brief PLL event
 * @param pointer to pll structure
 * @return None
 *
 * # PLL Algorithm #
 * PLL Algorithm consists of 3 main parts-
 * 1. Zero crossing (Zx) interrupt
 * 2. Frequency sync
 * 3. Phase sync
 *
 * ## Zero Crossing (Zx) interrupt ##
 * A Zero crossing detector circuit generates the reference square wave from grid that our
internally generated sine wave must
 * match in frequency as well as phase. The Zx circuit works in the following way:
 * - It generates a high to low for grid's positive zero crossing (0 degrees)
 * - It generates a low to high for grid's negative zero crossing (180 degrees)
 * We have set the capture module to trigger an interrupt on falling edge thus ensuring that we
get an interrupt everytime
 * grid crosses 0 degrees.
 * In the interrupt we do the following 2 things:
 * - We get the period of previous cycle in capture counts. This is used for FREQUENCY_SYNC.
Counts to frequency calculation
 * is given by the following formula:
 *  \f$ Grid_Frequency = (sysclk_freq_in_Hz) / (Capture_Counts) \f$
```

```
 * - We mark the phase difference of internally generated sine wrt reference wave. This is done by capturing the value of
 * theta_index at the ZX instant. Theta_index can vary from 0 to NO_OF_MINOR_CYCLES_PER_LINE_CYCLE-1 in one line cycle. Let us assume
 * a value of 512 for NO_OF_MINOR_CYCLES_PER_LINE_CYCLE. Thus theta_index varies from 0-511 in one line cycle. The following
 * table indicates phase releationship for different values of theta_index wrt to reference square wave.
 *
 * | Theta_Index @ | Phase      |
 * | ZX interrupt  |            |
 * |:-------------:|:----------:|
 * | 0             | Inphase    |
 * | 1-255         | Phase Lead |
 * | 257-511       | Phase Lag  |
 *
 * @ref phase_shift_index stores the value of theta_index at Zero Crossing
 *
 * ## PLL Event ##
 * PLL is a scheduled event that is executed once every line cycle. As mentioned earlier it includes -
 * FREQUENCY_SYNC and PHASE_SYNC. It also includes conditions to handle a few corner cases.
 *
 * ### FREQUENCY SYNC ###
 * Frequency syncing has only one step. Based on the capture_counts of the previous line cycle (determined from ZX interrupt),
 * we calculate what must be the next value written to Timer period so that 512 interrupts can be achieved in one line cycle.
 *
 *  That is all FREQUENCY_SYNC involves. To prevent abrupt frequency changes of our internal sine, we put a limit
 *  on maximum change in tbprd allowed in one cycle.
 *
 *  ### PHASE SYNC ###
 *  Phase syncing involves matching our internal sine wave to the reference signal in such a way that our theta_index
 *  is exactly 0 at ZX interrupt. As per the earlier table, we can determine whether we are leading or lagging wrt refer-
 *  ence wave. The phase correction action behaves according to following table:
 *
 *-------------------------------------------------
 * | Theta_Index @ | Phase      | Corrective    |
 * | ZX interrupt  |            | Action        |
 * |:-------------:|:----------:|:-------------:|
 * | 0             | Inphase    | No action     |
 * | 1-255         | Phase Lead | Increase tbprd|
 * | 257-511       | Phase Lag  | Reduce tbprd  |
 *-------------------------------------------------
 * As seen from the above table, if we are leading wrt reference signal, we sligtly reduce our frequency by increasing
 * timer period by tbprd_stepsize. If we are lagging, then we increase our frequency by reducing timer period by tbprd_stepsize.
 *
 * tbprd_stepsize is a variable that determines that by how many counts we need to go up or down to phase sync. A higher
 * value of stepsize allows faster syncing. We employ a higher stepsize when the phase shift between the reference signal
 * is greater than 5 (out of 512) counts. The table below determines tbprd_stepsize
 *
 *-----------------------------------
 * | Phase shift   | tbprd_stepsize |
```

```
 * |              |               |
 * |:-------------:|:-------------:|
 * | 0            | 0             | In phase
 * | 1-5          | 1             | Mildly out of phase
 * | >5           | 4             | Greatly out of phase
 *-------------------------------------
 *
 * ### Handling Corner cases ###
 *
 * In each of the below corner cases we move from whatever the current freqeuncy is, to nominal
frequency (50 or 60Hz).
 * These cases describe various failure conditions for pll.
 *
 * #### Grid Missing ####
 * When the grid is missing, no capture interrupts will be generated. In such a case we will hold
the previous timer period.
 *
 * #### Grid Frequency out of range ####
 * If the grid frequency is out of range then we will hold the previous timer period.
 */
void phase_locked_loop(pll_t *pll)
{
  int32_t systick_present, systick_next, systick_final, change;
  int32_t new_capture;
  uint16_t shift_index;

  /* Both next_tprd and present_tprd start with the present Timer register value */
  systick_next = systick_present = ROM_SysTickPeriodGet();
  shift_index = pll->phase_shift_index;

  if(pll->capture_detected == true)
  {
    /* Clear flag to allow next interrupts to set it  */
    pll->capture_detected = false;

    new_capture = pll->freq_in_cap_counts;
    systick_final = new_capture/SINE_SAMPLE_SIZE;

    /**
     * FREQUENCY SYNC
     * The change in tbprd allowed is restricted to MAX_INCREMENTAL_CHANGE_IN_COUNTS to
     * ensure a smooth transition to new frequency
     */
    change = _IQsat((systick_final - systick_present),MAX_INCREMENTAL_CHANGE, -
MAX_INCREMENTAL_CHANGE);
    systick_next = systick_present + change;

    /** PHASE SYNCING **/
    if(PHASE_LEAD(shift_index))
      systick_next++;
    else if(PHASE_LAG(shift_index))
      systick_next--;
  }

  if(systick_next < SYSTICK_LOW_LIMIT) // 3606
    systick_next = SYSTICK_LOW_LIMIT;
  else if(systick_next > SYSTICK_HIGH_LIMIT) // 4261
    systick_next = SYSTICK_LOW_LIMIT;
    /* update systick */
    ROM_SysTickPeriodSet(systick_next);
```

7. es_measurements.h

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_measurement.h
// header file foe measurement.c
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_LINE_MON_ES_MEASUREMENTS_H_
#define SRC_LINE_MON_ES_MEASUREMENTS_H_
#include "device.h"
#include "arm_math.h"
/************ Structure prototypes ***************************************/
/* rms structure */
typedef struct
{
    _iq norm_acc; /**< RMS accumulator */
    _iq norm_rms; /**< normalised rms value */
    _iq norm_offset; /**< Offset for rms calibration */
    _iq gain; /**< Gain for rms calibration */
} rms_struct_t;
/* metrics structure */
typedef struct
{
    rms_struct_t Iac;
    rms_struct_t Vac;
    _iq P_apparent;
    _iq P_active;
    _iq P_inst_acc;
    _iq P_reactive;
    _iq P_PowerFactor;
    _iq Phase_shift;
    _iq frequency;
    float32_t Vthd;
    float32_t Ithd;
    _iq V_Peak;
    _iq I_Peak;
} ac_metrics_t;

/************ Global Variables    ****************************************/
extern ac_metrics_t ac_metrics;
extern uint32_t ac_raw_adc_counts[2];

/************ macros *****************************************************/
#define ADC_LEVEL_SHIFT   (0.5)
#define _Q12toIQ24(A)     ((int32_t)A<<12)
#define V_FULL_SCALE  400.0
#define I_FULL_SCALE  49.5
#define P_FULL_SCALE  V_FULL_SCALE*I_FULL_SCALE
#define IQ24toFloat   16777216.0

/************ global functions ***************************************/
extern void init_adc();
#endif /* SRC_LINE_MON_ES_MEASUREMENTS_H_ */
```

8. es_measurements.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_measurement.c
// modules for AC_measurement
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/******************* Header files  ***************************************/
#include <es_measurements.h>
/*************************************************************************/

/****************** Global variables ***********************************/
ac_metrics_t ac_metrics;
uint32_t ac_raw_adc_counts[2];
/*************************************************************************/

/* desc: Initialize ADC_1 for voltage/current samples
 * args: none
 * ret : none
 */
void init_adc()
{
  /* SS0 triggers the following channels
   * PE0  - AIN3 - V_ac_sense
   * PE1  - AIN2 - I_ac_sense
   */

  /* Analog pins initialisation */
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
  ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0);
  ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_1);

  /* Initialise ADC */
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
  /* Sequencer 0, Software trigger, highest priority */
  ROM_ADCSequenceConfigure(ADC1_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
  /* Step 0 of Sequencer 0, CH3 */
  ROM_ADCSequenceStepConfigure(ADC1_BASE, 0, 0, ADC_CTL_CH3);
  /* Step 1 of Sequencer 0, CH2, End_channel, Generate interrupt */
  ROM_ADCSequenceStepConfigure(ADC1_BASE, 0, 1,
      ADC_CTL_CH2 | ADC_CTL_IE | ADC_CTL_END);
  /* Enable sequencer */
  ROM_ADCSequenceEnable(ADC1_BASE, 0);

  /* ac_metrics initialisation */
  ac_metrics.Iac.gain = _IQ(1.0);
  ac_metrics.Iac.norm_offset = _IQ(0);
  ac_metrics.Iac.norm_acc = 0;
  ac_metrics.Iac.norm_rms = 0;
  ac_metrics.Vac.gain = _IQ(1.0);
  ac_metrics.Vac.norm_offset = _IQ(0);
  ac_metrics.Vac.norm_acc = 0;
  ac_metrics.Vac.norm_rms = 0;
  ac_metrics.P_active = 0;
  ac_metrics.P_inst_acc = 0;
```

```
  ac_metrics.P_apparent = 0;
  ac_metrics.P_reactive = 0;
  ac_metrics.P_PowerFactor = 0;
  ac_metrics.V_Peak = 0;
  ac_metrics.I_Peak = 0;
}
```

9. es_gui.h

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_gui.h
// Header file for es_gui.c
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_DISPLAY_DISPLAY_OPERATIONS_H_
#define SRC_DISPLAY_DISPLAY_OPERATIONS_H_

/******************** Header files  ***************************************/
#include "../fft/es_fft_compute.h"
#include "../line_mon/es_measurements.h"
#include "../line_mon/es_pll.h"
#include "device.h"
#include "grlib/grlib.h"
#include "grlib/widget.h"
#include "grlib/canvas.h"
#include "grlib/pushbutton.h"
#include "drivers/Kentec320x240x16_ssd2119_SPI.h"
#include "drivers/touch.h"
/*****************************************************************************/

/******************** Global Variables ***************************************/
extern uint8_t g_uiSpectrum;
extern int8_t g_iPage;

/******************** Gui Pages ***************************************/
#define VOLTAGE_SPECTRUM  0
#define CURRENT_SPECTRUM  1
#define PAGE_METRICS            0
#define PAGE_VOLTAGE_SPECTRUM   1
#define PAGE_CURRENT_SPECTRUM   2
#define PAGE_TIME_DOMAIN_SIGNAL 3

#define LAST_PAGE     3

#define DISPLAY_SAMPLE_SIZE   256

/******************** Widgets ***************************************/
extern tPushButtonWidget g_sPushBtn;
extern tCanvasWidget g_sACMetricsWidget;
extern tCanvasWidget g_sVrms, g_sVrmsValue, g_sIrms, g_sIrmsValue, g_sFreq,
    g_sFreqValue, g_sPF, g_sPFValue, g_sP_apparent, g_sP_apparent_val,
    g_sP_active, g_sP_active_val, g_sTHDv, g_sTHDv_val, g_sTHDi, g_sTHDi_val,
    g_sPhase_val, g_sPhase,g_sVpeak,g_sIpeak,g_sVpeak_val,g_sIpeak_val;

/******************** Images ***************************************/
extern const uint8_t g_pui8Image[];
extern const uint8_t g_pui8Right24x23[];
extern const uint8_t g_pui8RightSmall15x14[];
extern const uint8_t g_pui8Left24x23[];
extern const uint8_t g_pui8LeftSmall15x14[];
```

```
/******************** Gui Context*********************************************/
extern tContext sContext;
extern tRectangle sRect;

/******************** Waveform control ***************************************/
extern _iq V_waveform_buffer[320];
extern _iq I_waveform_buffer[320];
extern uint16_t wave_index;
extern uint8_t skip_sample;

/******************** Global functions  **************************************/
extern void Display_FreqSpectrum(uint8_t param);
extern void display_setup_page(int8_t page_no);
extern void gui_init();
extern void Display_TimeDomain(void);

#endif /* SRC_DISPLAY_DISPLAY_OPERATIONS_H_ */
```

10.es_gui.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_gui.c
// Top level file that performs all the display update. This calls the underlying
// Graphics Library: http://www.ti.com/lit/ug/spmu300a/spmu300a.pdf
//
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/****************** Header files  ***************************************/
#include <es_gui.h>
/*****************************************************************************/

/****************** Global Variables ************************************/
tContext sContext;
tRectangle sRect;
uint8_t g_uiSpectrum = 0;
int8_t g_iPage = 0;
bool g_Led1On = false;
bool g_Led2On = false;
extern uint32_t sys_clk;
#define VOLTAGE_YAXIS    80
#define CURRENT_YAXIS    180

_iq V_waveform_buffer[320] =
{ 0 };
_iq I_waveform_buffer[320] =
{ 0 };
uint16_t wave_index = 0;
uint8_t skip_sample = 0;

/****************** Call back function for widgets **************************/
void OnNext(tWidget *pWidget);
void OnPrevious(tWidget *pWidget);
/*****************************************************************************/

/****************** Widgets *************************************************/
Canvas(g_sTitlePanel, 0, 0, 0, &g_sKentec320x240x16_SSD2119, 0, 0, 320, 24,
    CANVAS_STYLE_TEXT | CANVAS_STYLE_FILL, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm20b, "AC Metrics", 0, 0);

Canvas(g_sACMetricsWidget, 0, 0, &g_sVrms, &g_sKentec320x240x16_SSD2119, 0, 0,
    1, 1, 0, 0, 0, 0, 0, 0, 0, 0);

Canvas(g_sVrms, &g_sACMetricsWidget, 0, &g_sVrmsValue,
    &g_sKentec320x240x16_SSD2119, 0, 30, 60, 28,
    CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow, ClrCUYellow,
    &g_sFontCm16b, "Vrms", 0, 0);

Canvas(g_sVrmsValue, &g_sVrms, 0, &g_sIrms, &g_sKentec320x240x16_SSD2119, 55,
    30, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm18, "123.4 V", 0, 0);

Canvas(g_sIrms, &g_sVrmsValue, 0, &g_sIrmsValue, &g_sKentec320x240x16_SSD2119,
    0, 60, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
```

```
    ClrCUYellow, &g_sFontCm16b, "Irms", 0, 0);

Canvas(g_sIrmsValue, &g_sIrms, 0, &g_sFreq, &g_sKentec320x240x16_SSD2119, 55,
    60, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm18, "6.0  A", 0, 0);

Canvas(g_sFreq, &g_sIrmsValue, 0, &g_sFreqValue, &g_sKentec320x240x16_SSD2119,
    0, 90, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
    ClrCUYellow, &g_sFontCm16b, "Freq", 0, 0);

Canvas(g_sFreqValue, &g_sFreq, 0, &g_sPF, &g_sKentec320x240x16_SSD2119, 55, 90,
    60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm18, "59.9 Hz", 0, 0);

Canvas(g_sPF, &g_sFreqValue, 0, &g_sPFValue, &g_sKentec320x240x16_SSD2119, 0,
    120, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
    ClrCUYellow, &g_sFontCm16b, "PF  ", 0, 0);

Canvas(g_sPFValue, &g_sPF, 0, &g_sP_apparent, &g_sKentec320x240x16_SSD2119, 55,
    120, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "0.99", 0, 0);

Canvas(g_sP_apparent, &g_sPFValue, 0, &g_sP_apparent_val,
    &g_sKentec320x240x16_SSD2119, 204, 30, 60, 28,
    CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow, ClrCUYellow,
    &g_sFontCm16b, "P (VA)", 0, 0);

Canvas(g_sP_apparent_val, &g_sP_apparent, 0, &g_sP_active,
    &g_sKentec320x240x16_SSD2119, 259, 30, 60, 28,
    CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm18, "720.5", 0, 0);

Canvas(g_sP_active, &g_sP_apparent, 0, &g_sP_active_val,
    &g_sKentec320x240x16_SSD2119, 204, 60, 60, 28,
    CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow, ClrCUYellow,
    &g_sFontCm16b, "P (W)", 0, 0);

Canvas(g_sP_active_val, &g_sP_active, 0, &g_sTHDv, &g_sKentec320x240x16_SSD2119,
    259, 60, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "740.9", 0, 0);

Canvas(g_sTHDv, &g_sP_active_val, 0, &g_sTHDv_val, &g_sKentec320x240x16_SSD2119,
    0, 150, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
    ClrCUYellow, &g_sFontCm16b, "Thd(V)", 0, 0);

Canvas(g_sTHDv_val, &g_sTHDv, 0, &g_sTHDi, &g_sKentec320x240x16_SSD2119, 55,
    150, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "2.1 %", 0, 0);

Canvas(g_sTHDi, &g_sTHDv_val, 0, &g_sTHDi_val, &g_sKentec320x240x16_SSD2119, 0,
    180, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
    ClrCUYellow, &g_sFontCm16b, "Thd(I)", 0, 0);

Canvas(g_sTHDi_val, &g_sTHDi, 0, &g_sPhase, &g_sKentec320x240x16_SSD2119, 55,
    180, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "7.5 %", 0, 0);

Canvas(g_sPhase, &g_sTHDv_val, 0, &g_sPhase_val, &g_sKentec320x240x16_SSD2119,
    0, 210, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0, ClrCUYellow,
    ClrCUYellow, &g_sFontCm16b, "Phase", 0, 0);
```

```
Canvas(g_sPhase_val, &g_sPhase, 0, &g_sVpeak, &g_sKentec320x240x16_SSD2119, 55,
    210, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "7.5 %", 0, 0);

Canvas(g_sVpeak, &g_sPhase_val, 0, &g_sVpeak_val, &g_sKentec320x240x16_SSD2119,
    204, 180, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0,
    ClrCUYellow, ClrCUYellow, &g_sFontCm16b, "Vpk", 0, 0);

Canvas(g_sVpeak_val, &g_sVpeak, 0, &g_sIpeak, &g_sKentec320x240x16_SSD2119, 259,
    180, 60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0,
    ClrBlack, &g_sFontCm18, "155.5 V", 0, 0);

Canvas(g_sIpeak, &g_sVpeak_val, 0, &g_sIpeak_val, &g_sKentec320x240x16_SSD2119,
    204, 210, 60, 28, CANVAS_STYLE_TEXT | CANVAS_STYLE_TEXT_LEFT, 0,
    ClrCUYellow, ClrCUYellow, &g_sFontCm16b, "Ipk", 0, 0);

Canvas(g_sIpeak_val, &g_sIpeak, 0, 0, &g_sKentec320x240x16_SSD2119, 259, 210,
    60, 28, CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT, ClrCUYellow, 0, ClrBlack,
    &g_sFontCm18, "1.5 A", 0, 0);

/* Next button */
RectangularButton(g_sNext, 0, 0, 0, &g_sKentec320x240x16_SSD2119, 296, 0, 23,
    23, PB_STYLE_IMG, ClrCUYellow, ClrCUYellow, 0, 0, 0, 0, g_pui8Right24x23,
    g_pui8RightSmall15x14, 0, 0, OnNext);

/* Previous button */
RectangularButton(g_sPrevious, 0, 0, 0, &g_sKentec320x240x16_SSD2119, 0, 0, 23,
    23, PB_STYLE_IMG, ClrCUYellow, ClrCUYellow, 0, 0, 0, 0, g_pui8Left24x23,
    g_pui8LeftSmall15x14, 0, 0, OnPrevious);
/*****************************************************************************/

/* desc: Call back for next buttom pressed on display
 * args: pointer to widget
 * ret : none
 */
void OnNext(tWidget *pWidget)
{
  g_Led1On = !g_Led1On;
  if (g_Led1On)
  {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0xFF);
  }
  else
  {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0x00);
  }

  /* Page increment and wraparound */
  g_iPage++;
  if (g_iPage == LAST_PAGE + 1)
    g_iPage = 0;

  /* Setup the new page */
  display_setup_page(g_iPage);
}

/* desc: Call back for previous buttom pressed on display
 * args: pointer to widget
 * ret : none
 */
void OnPrevious(tWidget *pWidget)
```

```
{
  g_Led2On = !g_Led2On;
  if (g_Led2On)
  {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0xFF);
  }
  else
  {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0x00);
  }

  /* Page decrement and wraparound */
  g_iPage--;
  if (g_iPage == -1)
    g_iPage = LAST_PAGE;

  /* Setup the new page */
  display_setup_page(g_iPage);

}

/* desc: Initialize the display context. Dispaly the starting page
 * args: none
 * ret : none
 */
void gui_init()
{
  //
  // Initialize the display driver.
  //
  Kentec320x240x16_SSD2119Init();

  //
  // Initialize the graphics context.
  //
  GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);

  /* Draw logo */
  GrImageDraw(&sContext, g_pui8Image, 0, 0);
  GrFlush(&sContext);

  TouchScreenInit(sys_clk);
  TouchScreenCallbackSet(WidgetPointerMessage);

  WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sTitlePanel);
  WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sNext);
  WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sPrevious);
  WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sACMetricsWidget);
  WidgetPaint(WIDGET_ROOT);
}

/* desc: Setup the new page on a page change
 * args: page_no
 * ret : none
 */
void display_setup_page(int8_t page_no)
{
  if (page_no == PAGE_METRICS)
  {
    /* Draw logo */
    GrImageDraw(&sContext, g_pui8Image, 0, 0);
```

```
    GrFlush(&sContext);

    /* Change title text */
    CanvasTextSet(&g_sTitlePanel, "AC Metrics");

    /* Add the metrics canvas widget */
    WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sACMetricsWidget);
  }
  else if ((page_no == PAGE_VOLTAGE_SPECTRUM)
      || (page_no == PAGE_CURRENT_SPECTRUM))
  {
    /* fill screen with black. No logo */
    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = 319;
    sRect.i16YMax = 239;
    GrContextForegroundSet(&sContext, ClrBlack);
    GrRectFill(&sContext, &sRect);

    /* Add the metrics canvas widget */
    WidgetRemove((tWidget *) &g_sACMetricsWidget);

    /* Draw the frequency base */
    sRect.i16XMin = 0;
    sRect.i16YMin = 229;
    sRect.i16XMax = 319;
    sRect.i16YMax = 231;
    GrContextForegroundSet(&sContext, ClrCUYellow);
    GrRectFill(&sContext, &sRect);

    /* Change title text */
    if (page_no == PAGE_VOLTAGE_SPECTRUM)
    {
      CanvasTextSet(&g_sTitlePanel, "Voltage Spectrum");
      g_uiSpectrum = VOLTAGE_SPECTRUM;
    }
    else
    {
      CanvasTextSet(&g_sTitlePanel, "Current Spectrum");
      g_uiSpectrum = CURRENT_SPECTRUM;
    }
  }

  else if ((page_no == PAGE_TIME_DOMAIN_SIGNAL))
  {
    CanvasTextSet(&g_sTitlePanel, "Time Domain Signal");

    sRect.i16XMin = 0;
    sRect.i16YMin = 0;
    sRect.i16XMax = 319;
    sRect.i16YMax = 239;
    GrContextForegroundSet(&sContext, ClrBlack);
    GrRectFill(&sContext, &sRect);

    /* Add the metrics canvas widget */
    WidgetRemove((tWidget *) &g_sACMetricsWidget);
  }

  /* Draw the widgets */
  WidgetPaint(WIDGET_ROOT);
}
```

```c
/* desc: Display the frequency spectrum based on parameter
 * args: paramter (Voltage or Current)
 * ret : none
 */
void Display_FreqSpectrum(uint8_t param)
{
  /* We will use 20 bins to store 20 frequency amplitudes */
  int32_t frequency_bins[20];
  uint8_t i;

  /* Wipe the previous FFT */
  sRect.i16XMin = 0;
  sRect.i16YMin = 30;
  sRect.i16XMax = 319;
  sRect.i16YMax = 228;
  GrContextForegroundSet(&sContext, ClrBlack);

  GrRectFill(&sContext, &sRect);

  GrContextForegroundSet(&sContext, ClrCUYellow);

  if (param == VOLTAGE_SPECTRUM)
  {
    for (i = 0; i < 20; i++)
    {
      /* Scale the frequency bin values with the y axis span of the spectrum canvas */
      frequency_bins[i] = 229
          - (int32_t) (fft_output_array_voltage[i] * (240.0 - 40.0));
    }
  }
  else if (param == CURRENT_SPECTRUM)
  {
    for (i = 0; i < 20; i++)
    {
      /* Scale the frequency bin values with the y axis span of the spectrum canvas */
      frequency_bins[i] = 229
          - (int32_t) (fft_output_array_current[i] * 25 * (240.0 - 40.0));
    }
  }

  /* frequency bin now  contains y points */
  for (i = 0; i < 20; i++)
  {
    //GrLineDrawV(&sContext,i*16,229,frequency_bins[i]);
    GrLineDraw(&sContext, (i * 16 - 4), 229, i * 16, frequency_bins[i]);
    GrLineDraw(&sContext, (i * 16), frequency_bins[i], (i * 16 + 4), 229);
  }

  fft_output_array_voltage[0] = CURRENT_SPECTRUM;
}

/* desc: Display the time domain waveforms on the screen
 * args: none
 * ret : none
 */
void Display_TimeDomain(void)
{
  /* Declaring the variables */
  uint32_t XPixelCurrent = 0;
  uint32_t YPixelCurrent[DISPLAY_SAMPLE_SIZE];
```

```
  static uint32_t YPixelPrev_Volt[DISPLAY_SAMPLE_SIZE] =
  { 0 };
  static uint32_t YPixelPrev_Curr[DISPLAY_SAMPLE_SIZE] =
  { 0 };
  uint32_t DataCount = 0;


/* Writing Strings for Voltage and Current waveforms (Titles) */
  GrContextForegroundSet(&sContext, ClrCUYellow);
  GrContextFontSet(&sContext, &g_sFontCm16b);
  GrStringDrawCentered(&sContext, "Voltage", -1, 280, 31, 0);
  GrStringDrawCentered(&sContext, "Current", -1, 280, 131, 0);

/* Creating the X and Y axis */
  GrContextForegroundSet(&sContext, ClrWhite);
  GrLineDrawV(&sContext, 0, 30, 239);
  GrLineDrawH(&sContext, 0, 319, VOLTAGE_YAXIS);

/* Removing the previously plotted sine wave with black color. The background is black colour so
we can't see this line
/* Previously drawn time domain signal is removed by storing previous cycle's X and Y Pixels value
in static variable and
/* and drawing black line using GrLineDraw() function */

  /* Current time domain signal is drawn with yellow color by calculating the current values of X
and Y Pixel and scaling them appropriately */
  DataCount = 0;
  while (DataCount < DISPLAY_SAMPLE_SIZE)
  {
//    GrContextForegroundSet(&sContext, ClrBlack);
//    GrPixelDraw(&sContext,XPixelCurrent,YPixelPrev[DataCount]);
//    if(DataCount != 0)
    YPixelCurrent[DataCount] = VOLTAGE_YAXIS
        - VOLTAGE_YAXIS * ((V_waveform_buffer[DataCount]) / 16777216.0);
    if (DataCount != 0)
    {
      GrContextForegroundSet(&sContext, ClrBlack);
      /* Clear the previous cycle's data */
      if (YPixelPrev_Volt[DataCount] != 0)
      {
        GrLineDraw(&sContext, XPixelCurrent - 1, YPixelPrev_Volt[DataCount - 1],
             // Erasing previous signal //
            XPixelCurrent, YPixelPrev_Volt[DataCount]);
      }
      GrContextForegroundSet(&sContext, ClrCUYellow);
      GrLineDraw(&sContext, XPixelCurrent - 1, YPixelCurrent[DataCount - 1],
             // Plotting Current Signal //
          XPixelCurrent, YPixelCurrent[DataCount]);
    }
    DataCount++;
    XPixelCurrent++;
  }

  /* Storing Current Y Pixel value for removing the signal on next update */
  DataCount = 0;
  while (DataCount < DISPLAY_SAMPLE_SIZE)
  {
    YPixelPrev_Volt[DataCount] = YPixelCurrent[DataCount];
    DataCount++;
  }
```

```
  /* Repeating the same process for current waveform */
  GrContextForegroundSet(&sContext, ClrWhite);
  GrLineDrawH(&sContext, 0, 319, CURRENT_YAXIS);

  DataCount = 0;
  XPixelCurrent = 0;
  while (DataCount < DISPLAY_SAMPLE_SIZE)
  {
//    GrContextForegroundSet(&sContext, ClrBlack);
//    GrPixelDraw(&sContext,XPixelCurrent,YPixelPrev[DataCount]);
//    if(DataCount != 0)
    YPixelCurrent[DataCount] = CURRENT_YAXIS
        - CURRENT_YAXIS * ((I_waveform_buffer[DataCount]) * 6 / 16777216.0);
    if (DataCount != 0)
    {
      GrContextForegroundSet(&sContext, ClrBlack);
      /* Clear the previous cycle's data */
      if (YPixelPrev_Curr[DataCount] != 0)
      {
        GrLineDraw(&sContext, XPixelCurrent - 1, YPixelPrev_Curr[DataCount - 1],
            XPixelCurrent, YPixelPrev_Curr[DataCount]);
      }
      GrContextForegroundSet(&sContext, ClrCUYellow);
      GrLineDraw(&sContext, XPixelCurrent - 1, YPixelCurrent[DataCount - 1],
          XPixelCurrent, YPixelCurrent[DataCount]);
    }
    DataCount++;
    XPixelCurrent++;
  }

  DataCount = 0;
  while (DataCount < DISPLAY_SAMPLE_SIZE)
  {
    YPixelPrev_Curr[DataCount] = YPixelCurrent[DataCount];
    DataCount++;
  }
}
```

11. es_enet_datalogger.h

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_enet_datalogger.h
// Header file for es_enet_datalogger.c
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

#ifndef SRC_ENET_ES_ENET_DATALOGGER_H_
#define SRC_ENET_ES_ENET_DATALOGGER_H_

/******************** Header files  ***************************************/
#include "utils/lwiplib.h"
/*************************************************************************/

//*****************************************************************************
//
// Defines for setting up the system clock.
//
//*****************************************************************************
#define SYSTICKHZ               100
#define SYSTICKMS               (1000 / SYSTICKHZ)

//*****************************************************************************
//
// Interrupt priority definitions.  The top 3 bits of these values are
// significant with lower values indicating higher priority interrupts.
//
//*****************************************************************************
#define SYSTICK_INT_PRIORITY    0x80
#define ETHERNET_INT_PRIORITY   0xC0

#define ENET_EVENT_COUNTER_ELAPSED  65
/*************************************************************************/
extern uint8_t nw_update_timer;

/****************** Global variables ************************************/
extern void enet_init();
extern void enet_server_connect();
extern void enet_metrics_log();

#endif /* SRC_ENET_ES_ENET_DATALOGGER_H_ */
```

12. es_enet_datalogger.c

```
//*****************************************************************************
// Compact Analyser for Power Equipment (CAPE)
// es_enet_datalogger.c
// Ethernet datalogger. Uses LwIP stack
// LwIP stack: http://savannah.nongnu.org/projects/lwip/
// LwIP docs: http://lwip.wikia.com/wiki/LwIP_Application_Developers_Manual
//
// Author
// Meher Jain
// Vivek Sankaranarayanan
//*****************************************************************************

/******************* Header files  ***************************************/
#include <es_enet_datalogger.h>
#include "device.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include <string.h>
//#include "driverlib/rom_map.h"
/*************************************************************************/

#define FAULT_SYSTICK           15          // System Tick

extern uint32_t sys_clk;
uint32_t g_ui32IPAddress;
uint8_t nw_update_timer = 0;

extern char display_update_buffer1[10];
extern char display_update_buffer2[10];
extern char display_update_buffer3[10];
extern char display_update_buffer4[10];
extern char display_update_buffer5[10];
extern char display_update_buffer6[10];
extern char display_update_buffer7[10];
extern char display_update_buffer8[10];
extern char display_update_buffer9[10];
extern char display_update_buffer10[10];
extern char display_update_buffer11[10];
volatile bool bPreviousDataTransmitted = true;
volatile bool bConnectedToServer = false;

struct tcp_pcb *pcb;
/***************************** User Callbacks ******************************/
/* desc: Callback for handling tcp connect error
 * args: Dummy argument, error
 * ret : none
 */
void OnTcpError(void *arg, err_t err)
{
  UARTprintf("Error:%d", err);
  while (1)
  {
  };
}

/* desc: On a successful transmit
 * args: Dummy argument, pointer to protocol control block, lenth of bytes transmitted
 * ret : none
```

```
 */
err_t OnSent(void *arg, struct tcp_pcb *tpcb, u16_t len)
{
  bPreviousDataTransmitted = 1;
  UARTprintf("Booyah!!\n\r");
  return ERR_OK;
}

/* desc: On a successful connect
 * args: Dummy argument, pointer to protocol control block, error code
 * ret : none
 */
err_t OnClientConnected(void *arg, struct tcp_pcb *pcb, err_t err)
{
  LWIP_UNUSED_ARG(arg);
  if (err == ERR_OK)
  {
    UARTprintf("Connection Success\n\r");
    tcp_sent(pcb, OnSent);
    bConnectedToServer = true;
  }
  return err;
}

/***************************** End of Callbacks *****************************/

/* desc: Log AC metrics over ethernet
 * args: none
 * ret : none
 */
void enet_metrics_log()
{

  char enet_update_buffer[200] =
  { 0 };

//  UARTprintf("%s\n\r",enet_update_buffer);

  if (bPreviousDataTransmitted)
  {
    sprintf(enet_update_buffer, "%s$%s$%s$%s$%s$%s$%s$%s$%s$%s$%s$\n\r",
        display_update_buffer1, display_update_buffer2, display_update_buffer3,
        display_update_buffer4, display_update_buffer5, display_update_buffer6,
        display_update_buffer7, display_update_buffer8, display_update_buffer9,
        display_update_buffer10, display_update_buffer11);
    bPreviousDataTransmitted = 0;

    // TODO: Check buffer remaining using tcp_sndbuf
    tcp_write(pcb, (const void *) enet_update_buffer,
        strlen(enet_update_buffer), 0);
    tcp_output(pcb);
  }
  else
  {
    UARTprintf("Previous data still not transmitted\n\r");
  }
}

/* desc: Attempt server connection
 * args: none
 * ret : none
```

```
 */
void enet_server_connect()
{
  ip_addr_t remote_addr;

  pcb = tcp_new();
  tcp_arg(pcb, pcb);
  tcp_err(pcb, OnTcpError);

  /* Connect */
  IP4_ADDR(&remote_addr, 192, 168, 1, 2);
  tcp_connect(pcb, &remote_addr, 5000, OnClientConnected);
  while (!bConnectedToServer)
  {
  }

  /* Send test data */
  tcp_write(pcb, "$\n\r", strlen("$\n\r"), 0);
  tcp_output(pcb);
}

/* desc: Initialise Ethernet environment
 * args: none
 * ret : none
 */
void enet_init()
{
  /* from enet_lwip prj */
  uint32_t ui32User0, ui32User1;
  uint8_t pui8MACArray[8];

  //
  // Configure the hardware MAC address for Ethernet Controller filtering of
  // incoming packets.  The MAC address will be stored in the non-volatile
  // USER0 and USER1 registers.
  //
  ROM_FlashUserGet(&ui32User0, &ui32User1);
  if ((ui32User0 == 0xffffffff) || (ui32User1 == 0xffffffff))
  {
    //
    // We should never get here.  This is an error if the MAC address has
    // not been programmed into the device.  Exit the program.
    // Let the user know there is no MAC address
    //
    UARTprintf("No MAC programmed!\n");
    while (1)
    {
    }
  }
  //
  // Convert the 24/24 split MAC address from NV ram into a 32/16 split MAC
  // address needed to program the hardware registers, then program the MAC
  // address into the Ethernet Controller registers.
  //
  pui8MACArray[0] = ((ui32User0 >> 0) & 0xff);
  pui8MACArray[1] = ((ui32User0 >> 8) & 0xff);
  pui8MACArray[2] = ((ui32User0 >> 16) & 0xff);
  pui8MACArray[3] = ((ui32User1 >> 0) & 0xff);
  pui8MACArray[4] = ((ui32User1 >> 8) & 0xff);
  pui8MACArray[5] = ((ui32User1 >> 16) & 0xff);
```

```
  //
  // Initialize the lwIP library, using DHCP.
  //
  uint32_t ipaddr;
  uint32_t netmask;
  uint32_t gateway;
  ipaddr = (192 << 24) | (168 << 16) | (1 << 8) | (3);
  netmask = (255 << 24) | (255 << 16) | (255 << 8) | (0);
  gateway = (192 << 24) | (168 << 16) | (1 << 8) | (1);
  lwIPInit(sys_clk, pui8MACArray, ipaddr, netmask, gateway,
  IPADDR_USE_STATIC);

  //
  // Set the interrupt priorities.  We set the SysTick interrupt to a higher
  // priority than the Ethernet interrupt to ensure that the file system
  // tick is processed if SysTick occurs while the Ethernet handler is being
  // processed.  This is very likely since all the TCP/IP and HTTP work is
  // done in the context of the Ethernet interrupt.
  //
  ROM_IntPrioritySet(INT_EMAC0, ETHERNET_INT_PRIORITY);
  ROM_IntPrioritySet(FAULT_SYSTICK, SYSTICK_INT_PRIORITY);
}

//*****************************************************************************
//
// Display an lwIP type IP Address.
//
//*****************************************************************************
void DisplayIPAddress(uint32_t ui32Addr)
{
  char pcBuf[16];

  //
  // Convert the IP Address into a string.
  //
  usprintf(pcBuf, "%d.%d.%d.%d", ui32Addr & 0xff, (ui32Addr >> 8) & 0xff,
      (ui32Addr >> 16) & 0xff, (ui32Addr >> 24) & 0xff);

  //
  // Display the string.
  //
  UARTprintf(pcBuf);
}

//*****************************************************************************
//
// Required by lwIP library to support any host-related timer functions.
//
//*****************************************************************************
void lwIPHostTimerHandler(void)
{
  uint32_t ui32Idx, ui32NewIPAddress;

  //
  // Get the current IP address.
  //
  ui32NewIPAddress = lwIPLocalIPAddrGet();

  //
  // See if the IP address has changed.
  //
```

```
  if (ui32NewIPAddress != g_ui32IPAddress)
  {
    //
    // See if there is an IP address assigned.
    //
    if (ui32NewIPAddress == 0xffffffff)
    {
      //
      // Indicate that there is no link.
      //
      UARTprintf("Waiting for link.\n");
    }
    else if (ui32NewIPAddress == 0)
    {
      //
      // There is no IP address, so indicate that the DHCP process is
      // running.
      //
      UARTprintf("Waiting for IP address.\n");
    }
    else
    {
      //
      // Display the new IP address.
      //
      UARTprintf("IP Address: ");
      DisplayIPAddress(ui32NewIPAddress);
      UARTprintf("\nOpen a browser and enter the IP address.\n");
    }

    // Save the new IP address.

    g_ui32IPAddress = ui32NewIPAddress;


  }
  //
  // If there is not an IP address.
  //
  if ((ui32NewIPAddress == 0) || (ui32NewIPAddress == 0xffffffff))
  {
    //
    // Loop through the LED animation.
    //

    for (ui32Idx = 1; ui32Idx < 17; ui32Idx++)
    {
      SysCtlDelay(sys_clk / (ui32Idx << 1));
    }
  }
}
```

13. remote_server.c

```
/*
Author:
Meher Jain
Vivek Sankaranarayanan

-This C script creates a server based on TCP/IP for CAPE to log the measured data over ethernet.
-CAPE sends the complete AC Metric with 11 elements every 1.08seconds. It uses '$' as delimiter
between every metric elemetns
-Server being aware of the fact copies the received data in to a csv file.

-Server is running on following Specification:

1. IP ADDRESS - 192.168.1.3
2. Port No    - 5000
*/

/******************************** Header Files ***************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
#include <arpa/inet.h>
#include <time.h>
/*************************************************************************************/

/********************************** Macros *******************************************/
#define BUFFERLENGTH 512
#define PORTNO              5000
/*************************************************************************************/

/*
   Main File.
   This file creates a TCP/IP based server on specified IP address and port number. The server
   creates a TCP/IP socket and waits for client to connect into. After accepting the client
   connection, it sits in an infinite loop and logs the data sent by the client.
   args: None
   returns: None
*/

void main (int argc, char* argv[])
{
       int sock,newsocket;
       char currtime[50] = "\0";
       int i;
       const char *filelog = "ac_metric_log.csv";

       struct sockaddr_in serverAddr; /* Server addr */
       struct sockaddr_in clientAddr; /* Client addr */
       char receive_buffer[BUFFERLENGTH] = {'\0'};

       FILE *fp1;                                          // Declaring the file pointer //
       int clientlen,count =0;
       time_t t;                                           // Time structure //

       fp1 = fopen(filelog,"w");                           // Opening File for logging //

       if(fp1 == NULL)
              printf("ERROR in Opening log file\n");
```

```
	/* Hardcoding the title and subtitile for every element sent by the CAPE. It sends the data
in the same order as 			subtitles written to the file like 1. VRMS, 2. IRMS, 3.
Freq, etc.*/
	fprintf(fp1,"\t\t\t\tAC METRICS\n");
	fprintf(fp1,"Time$VRMS$IRMS$Frequency$PowerFactor$Apparent Power(VA)$Active
Power(W)$THD(V)$THI(A)$Phase$Vpeak$Ipeak");
	//fprintf(fp1,"\r\nHello");
	fflush(fp1);				// Flusing the local buffer in to the file //



	/* Opening a socket for TCP/IP Communication */

	if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1 )
	{
	   printf("ERROR: Failed to obtain Socket Descriptor");
	   exit(1);
	}
	else
	   printf("[Server] Obtaining socket descriptor successfully.\n");

	/* Fill the client socket address struct */
	serverAddr.sin_family = AF_INET; // Protocol Family
	serverAddr.sin_port = htons(PORTNO); // Port number
	serverAddr.sin_addr.s_addr = inet_addr("192.168.1.2"); // Fill the local ethernet address
	bzero(&(serverAddr.sin_zero), 8); // Flush the rest of struct


	/* Bind a special Port */
	if( bind(sock, (struct sockaddr*)&serverAddr, sizeof(struct sockaddr)) == -1 )
	{
	   printf("ERROR: Failed to bind Port.\n");
	   exit(1);
	}
	else
	  printf ("Binded the port 5000 ....\n");


	/* Listen remote connect/calling */
	if(listen(sock,10) == -1)
	{
	   printf("ERROR: Failed to listen Port.\n");
	   exit(1);
	}
	else
	   printf ("Listening on the port 5000 ....\n");


	clientlen = sizeof(struct sockaddr_in);	// Size of socket structure to fillout the
client address//


	/* Accepting the connection (Blocking Function), waits till there is a a connect request
from client */
	newsocket = accept(sock, (struct sockaddr *) &clientAddr, &clientlen);
	if (newsocket == -1) printf("Error Accepting the connection");
		printf("Connection Accepted....\n");
```

```
	/* Stay in infinite loop after client is connected for logging the data */
	while(1)
	{
		// For receiving the data from client//
		int rcvd = recv(newsocket, receive_buffer,sizeof receive_buffer ,0);
		/* If no data received from client, exit the server */
		if (rcvd <= 0)
		{
			printf("No Data received from client, Exiting from the child Sever...\n");
			close(newsocket);
			exit(0);
		}

		/* Finding current date & time for logging purpose */
		time(&t);
		strncpy(currtime,ctime(&t),(strlen(ctime(&t))-1)); // Copying the result to a buffer
//

		// Not copying the time for the first ACK received //

		if(strcmp(receive_buffer,"$\n\r"))                    {
			fprintf(fp1,"%s$",currtime);
		}

		fprintf(fp1,"%s\n",receive_buffer);     // copying received data in to the file //

		fflush(fp1);               // Flushing the local buffer in to the file pointer//
		printf("Data Logged...\n");


	}

	close(newsocket);          // Closing the socket // (It never reaches here)
}
```