# Digital Energy Meter

arosspope

June 11, 2017

# Contents

# 1 main.c

```
/* ###################################################################
**     Filename    : main.c
**     Project     : Digital Energy Meter
**     Processor   : MK70FN1M0VMJ12
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2015-07-20, 13:27, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file main.c
** @version 01.0
** @brief
**         Main module.
**         This module contains user's application code.
*/
/*!
**  @addtogroup main_module main module documentation
**  @{
*/
/* MODULE main */
#include "Cpu.h"
#include "Events.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "meter.h"
#include "interface.h"
#include "OS.h"

#define THREAD_STACK_SIZE 400

static uint32_t InterfacePacketStack[THREAD_STACK_SIZE] __attribute__ ((aligned
    (0x08)));
static uint32_t InterfacePrintStack[THREAD_STACK_SIZE]  __attribute__ ((aligned
    (0x08)));
static uint32_t InterfaceCycleStack[THREAD_STACK_SIZE]  __attribute__ ((aligned
    (0x08)));
static uint32_t MeterCalcStack[THREAD_STACK_SIZE]       __attribute__ ((aligned
    (0x08)));
static uint32_t MeterSampleStack[THREAD_STACK_SIZE]     __attribute__ ((aligned
    (0x08)));

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  BOOL initOk = bFALSE;
```

```
  OS_ERROR error;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  OS_DisableInterrupts();

  initOk = Interface_Init() && Meter_Init();  //Initialize the DEM
  OS_Init(CPU_CORE_CLK_HZ);                   //Initialize the RTOS

  //Create Threads
  OS_ThreadCreate(Interface_PacketHandle,
                  NULL,
                  &InterfacePacketStack[THREAD_STACK_SIZE - 1],
                  4);

  OS_ThreadCreate(Interface_CycleDisplay,
                  NULL,
                  &InterfaceCycleStack[THREAD_STACK_SIZE - 1],
                  3);

  OS_ThreadCreate(Interface_PrintData,
                  NULL,
                  &InterfacePrintStack[THREAD_STACK_SIZE - 1],
                  2);

  OS_ThreadCreate(Meter_RunCalculations,
                  NULL,
                  &MeterCalcStack[THREAD_STACK_SIZE - 1],
                  1);

  OS_ThreadCreate(Meter_GetSamples,
                  NULL,
                  &MeterSampleStack[THREAD_STACK_SIZE - 1],
                  0);

  if (initOk)
  {
    OS_Start();   //Start Multi-threading!
  }

  OS_EnableInterrupts();
  for(;;){}
  /*** Don't write any code pass this line, or it will be deleted during code
      generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component.
       DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is
          defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
/* END main */
```

```
/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

## 2 DAC.c

```
/*! @file
 *
 *  @brief Digital to Analog Converter.
 *
 *  This module contains routines that allows a caller to obtain data
     information about a waveform.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup DAC_module DAC module documentation
**  @{
*/
/* MODULE DAC */
#include "DAC.h"

/*!< Simple LUT of a sine wave, ranging from angle 0 to 360 in 5.625 degree
    steps. The sampled values in 32Q16 format. */
const int32_t const SineWave[65] = {0, 6423, 12785, 19024, 25079, 30893, 36409,
    41575, 46340, 50660, 54491, 57797,
                                    60547, 62714, 64276, 65220, 65536, 65220,
                                        64276, 62714, 60547, 57797, 54491,
                                    50660, 46340, 41575, 36409, 30893, 25079,
                                        19024, 12785, 6423, 0, -6423, -12785,
                                    -19024, -25079, -30893, -36409, -41575,
                                        -46340, -50660, -54491, -57797, -60547,
                                    -62714, -64276, -65220, -65536, -65220,
                                        -64276, -62714, -60547, -57797, -54491,
                                    -50660, -46340, -41575, -36409, -30893,
                                        -25079, -19024, -12785, -6423, 0};

typedef struct
{
  int16_t Amplitude;
}TWaveForm;

static TWaveForm VoltageWave;   /*!< Voltage Waveform. Amplitude base: 1000/2^15
     */
static TWaveForm CurrentWave;   /*!< Current Waveform. Amplitude base: 10/2^15
    */
static uint8_t PhaseDifference; /*!< The phase difference of the current
    waveform in reference to voltage. (Simply an integer offset within the array)
    . */

BOOL DAC_Init(int16_t voltageAmp, int16_t currentAmp, uint16_t phaseStep)
{
  VoltageWave.Amplitude = voltageAmp;
  CurrentWave.Amplitude = currentAmp;
  DAC_SetPhaseDifference(phaseStep);

  return (bTRUE);
}

/*! @brief Will pick out a value from the Sine LUT.
```

```
 *
 *  @param offset   - Place to pick sine value out of array (0 - 64)
 *  @return int32_t - Value at LUT[offset]
 */
int32_t getValueFromLUT(uint16_t offset)
{
  uint16_t tempTheta;

  if (offset >= 0 && offset <= 64)
  {
    return SineWave[offset];
  }
  else
  {
    return 0;   //In the unlikely event that the user reads at an invalid offset
        , we just return 0.
  }
}

/*! @brief Get a sample from a 'self-test' waveform.
 *
 *  @param sample - A pointer to a variable to hold the sample.
 *  @param offset - Where to 'pick-off' the variable from the wave.
 *  @param waveform - The type of waveform
 */
void sampleWaveform(int16_t * sample, uint16_t offset, TWaveForm waveform)
{
  int64_t result = getValueFromLUT(offset) * waveform.Amplitude ;
  *sample = (int16_t)(result / 65536);  //Scale value back down to original base
       of waveform
}

uint32_t DAC_PowerFactor(void)
{
  return ((uint32_t) getValueFromLUT((16 + PhaseDifference) % 64));
}

void DAC_GetSample(int16_t * voltSample, int16_t * currSample, uint16_t voltPos,
     uint16_t currPos)
{
  //Ensure that voltPos and currPos are within the bounds of 64 size array
  voltPos = (voltPos % 64);
  currPos = ((currPos + PhaseDifference) % 64);  //Some phase difference may
       occur between the two waves (current in reference to voltage)

  sampleWaveform(voltSample, voltPos, VoltageWave);
  sampleWaveform(currSample, currPos, CurrentWave);
}

void DAC_SetVoltageAmplitude(int16_t voltageAmp)
{
  VoltageWave.Amplitude = voltageAmp;
}

void DAC_SetCurrentAmplitude(int16_t currentAmp)
{
  CurrentWave.Amplitude = currentAmp;
```

```
}

void DAC_SetPhaseDifference(uint8_t phaseStep)
{
  /* Phase is represented as a series of steps to the outside world
   * i.e. phaseStep = 0  -> -90 deg
   *      phaseStep = 16 -> 0 deg
   *      phaseStep = 24 -> 45 deg
   */
  PhaseDifference = (48 + phaseStep) % 64;
}
/* END DAC */
/*!
** @}
*/
```

# 3 DAC.h

```
/*! @file
 *
 *  @brief Digital to Analog Converter.
 *
 *  This module contains routines that allows a caller to obtain data
     information about a waveform.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef DAC_H
#define DAC_H

#include "types.h"


/*! @brief Initializes the DAC before first use.
 *
 *  @param voltageAmp - Will set the Amplitude of the voltage waveform.
 *  @param currentAmp - Will set the Amplitude of the current waveform.
 *  @param phaseStep  - The phase difference of the current waveform in
     reference to voltage. (0 - 32)
 *  @return BOOL      - TRUE if initialization was successful.
 *  @note Assumes voltageAmp & currentAmp have already been normalized to their
     respective 'bases'.
 */
BOOL DAC_Init(int16_t voltageAmp, int16_t currentAmp, uint16_t phaseStep);

/*! @brief Returns the power factor based on the phase difference.
 *
 *  @return uint32_t - power factor (base 2^16)
 */
uint32_t DAC_PowerFactor(void);

/*! @brief Obtains a voltage and current sample of corresponding 'waveforms'.
 *
 *  @param voltSample - Pointer to a variable to store the voltage sample
 *  @param currSample - Pointer to a variable to store the current sample
 *  @param voltPos    - Where to sample within the LUT.
 *  @param currPos    - Where to sample within the LUT.
 *  @return void
 */
void DAC_GetSample(int16_t * voltSample, int16_t * currSample, uint16_t voltPos,
     uint16_t currPos);

/*! @brief Sets the amplitude of the voltage waveform.
 *
 *  @param voltageAmp - The amplitude to set. (32Q16)
 *  @return void
 */
void DAC_SetVoltageAmplitude(int16_t voltageAmp);

/*! @brief Sets the amplitude of the current waveform.
 *
 *  @param currentAmp - The amplitude to set. (32Q16)
```

```
 *  @return void
 */
void DAC_SetCurrentAmplitude(int16_t currentAmp);

/*! @brief Sets the phase difference between current in reference to voltage
 *
 *  @param phaseDifference - The phase step difference to set.
 *  @return void
 */
void DAC_SetPhaseDifference(uint8_t phaseStep);
#endif
```

# 4 Events.c

```
/* ###################################################################
**     Filename    : Events.c
**     Project     : Lab2
**     Processor   : MK70FN1M0VMJ12
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2015-08-17, 11:05, # CodeGen: 8
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file Events.c
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"

#ifdef __cplusplus
extern "C" {
#endif


/* User includes (#include below this line is not maintained by Processor Expert
    ) */

/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
```

```
*/
```

# 5 Events.h

```
/* ###################################################################
**     Filename    : Events.h
**     Project     : Lab2
**     Processor   : MK70FN1M0VMJ12
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2015-08-17, 11:05, # CodeGen: 8
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file Events.h
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/

#ifndef __Events_H
#define __Events_H
/* MODULE Events */

#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#ifdef __cplusplus
extern "C" {
#endif


/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

#endif
/* ifndef __Events_H*/
/*!
** @}
*/
/*
** ###################################################################
**
```

```
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

# 6 FIFO.c

```
/*! @file
 *
 *  @brief Routines to implement a FIFO buffer.
 *
 *  This contains the structure and "methods" for accessing a byte-wide FIFO.
 *
 *  @author BAllen, APope
 *  @date 2015-08-15
 */
/*!
**  @addtogroup FIFO_module FIFO module documentation
**  @{
*/
/* MODULE FIFO */
#include "Cpu.h"
#include "FIFO.h"
#include "OS.h"

void FIFO_Init(TFIFO * const FIFO)
{
  FIFO->Start = 0;
  FIFO->End = 0;
  FIFO->NbBytes = 0;
}

BOOL FIFO_Put(TFIFO * const FIFO, const uint8_t data)
{
  OS_DisableInterrupts();                       //Enter Critical Section

  if (FIFO->NbBytes < FIFO_SIZE)                //Ensure that the buffer is not
      full before putting data within it
  {
    FIFO->NbBytes++;                                                  //Increment
        data counter
    FIFO->Buffer[FIFO->End] = data;                     //Store data in the buffer
    FIFO->End = (FIFO->End + 1) % FIFO_SIZE;     //increment end pointer within
        FIFO_SIZE

    OS_EnableInterrupts();
    return (bTRUE);
  }

  OS_EnableInterrupts();
  return (bFALSE);
}

BOOL FIFO_Get(TFIFO * const FIFO, uint8_t * const dataPtr)
{
  OS_DisableInterrupts();                           //Enter Critical Section

  if (FIFO->NbBytes != 0)
  {
    FIFO->NbBytes--;
    *dataPtr = FIFO->Buffer[FIFO->Start];
    FIFO->Start = (FIFO->Start + 1) % FIFO_SIZE;
```

```
    OS_EnableInterrupts();
    return (bTRUE);
  }

  OS_EnableInterrupts();
  return (bFALSE);
}
/* END FIFO */
/*!
** @}
*/
```

# 7 FIFO.h

```
/*! @file
 *
 *  @brief Routines to implement a FIFO buffer.
 *
 *  This contains the structure and "methods" for accessing a byte-wide FIFO.
 *
 *  @author PMcL
 *  @date 2015-07-23
 */

#ifndef FIFO_H
#define FIFO_H

// new types
#include "types.h"

// Number of bytes in a FIFO
#define FIFO_SIZE 256

// The TFIFO structure which is used to implement a general-purpose byte-sized
    first-in/last-out data structure.
/*!
 * @struct TFIFO
 */
typedef struct
{
  uint16_t Start;               /*!< The index of the position of the oldest
      data in the FIFO */
  uint16_t End;                 /*!< The index of the next available empty
      position in the FIFO */
  uint16_t volatile NbBytes;    /*!< The number of bytes currently stored in the
       FIFO */
  uint8_t Buffer[FIFO_SIZE];    /*!< The actual array of bytes to store the data
       */
} TFIFO;

/*! @brief Initialize the FIFO before first use.
 *
 *  @param FIFO A pointer to the FIFO that needs initializing.
 *  @return void
 */
void FIFO_Init(TFIFO * const FIFO);

/*! @brief Put one character into the FIFO.
 *
 *  @param FIFO A pointer to a FIFO struct where data is to be stored.
 *  @param data A byte of data to store in the FIFO buffer.
 *  @return BOOL - TRUE if data is successfully stored in the FIFO.
 *  @note Assumes that FIFO_Init has been called.
 */
BOOL FIFO_Put(TFIFO * const FIFO, const uint8_t data);

/*! @brief Get one character from the FIFO.
 *
 *  @param FIFO A pointer to a FIFO struct with data to be retrieved.
```

```
 *  @param dataPtr A pointer to a memory location to place the retrieved byte.
 *  @return BOOL - TRUE if data is successfully retrieved from the FIFO.
 *  @note Assumes that FIFO_Init has been called.
 */
BOOL FIFO_Get(TFIFO * const FIFO, uint8_t * const dataPtr);

#endif
```

# 8 FTM.c

```
/*! @file
 *
 *  @brief Routines for setting up the flexible timer module (FTM) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the flexible timer module (FTM).
 *
 *  @author BAllen, APope
 *  @date 2015-09-04
 */
/*!
**  @addtogroup FTM_module FTM module documentation
**  @{
*/
/* MODULE FTM */
#include "MK70F12.h"
#include "FTM.h"
#include "OS.h"

#define CLOCK_SRC_NONE              0
#define CLOCK_SRC_SYSTEM            1
#define CLOCK_SRC_FIXED_FREQ    2
#define CLOCK_SRC_EXTERNAL        3

#define PRESCALE_DIVIDE_1           0
#define PRESCALE_DIVIDE_2           1
#define PRESCALE_DIVIDE_4           2
#define PRESCALE_DIVIDE_8           3
#define PRESCALE_DIVIDE_16        4
#define PRESCALE_DIVIDE_32        5
#define PRESCALE_DIVIDE_64        6
#define PRESCALE_DIVIDE_128       7

#define NUM_FTM_CHANNELS            8

/*!
 * @struct ChannelCallBack
 */
typedef struct
{
  TTimerFunction timerFunction;         /*!< Select function of timer channel (
      Input Capture | Output Compare)*/
  void (*userFunction)(void *);         /*!< Callback function to run in timer
      ISR*/
  void *userArguments;                          /*!< Arguments to pass into
      callback function*/
} ChannelCallBack;

static ChannelCallBack FTM0Channels[NUM_FTM_CHANNELS];  /*!< Table of callback
    information for each channel */

BOOL FTM_Init()
{
  //Enable Clock gating
  SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;
```

```
  //Initially load the Modulo and Count Registers
  FTM0_CNTIN = 0;
  FTM0_MOD = 0xFFFF;
  FTM0_CNT = 0;

  //Set-up Status and Control register
  FTM0_SC &= ~FTM_SC_TOIE_MASK;                          //Timer Overflow Interrupt
       Enable: 0 disabled
  FTM0_SC &= ~FTM_SC_CPWMS_MASK;                         //Center-Aligned PWM
      Select: 0 Up counting mode

  FTM0_SC &= ~FTM_SC_CLKS_MASK;                          //Clock Source Selection
  FTM0_SC |= FTM_SC_CLKS(CLOCK_SRC_FIXED_FREQ);
  FTM0_SC &= ~FTM_SC_PS_MASK;                             //Pre-scale Factor
      Selection:
  FTM0_SC |= FTM_SC_PS(PRESCALE_DIVIDE_1);

  //Init NVIC
  NVICICPR1 = (1 << 30);
  NVICISER1 = (1 << 30);

  FTM0_MODE |= FTM_MODE_FTMEN_MASK;

  return (bTRUE);
}

BOOL FTM_Set(const TFTMChannel* const aFTMChannel)
{
  //Check that the channel is within the known range (0 - 7)
  if (aFTMChannel->channelNb >= 0 && aFTMChannel->channelNb < NUM_FTM_CHANNELS)
  {
    //Init the FTM0 Status & Control register for the particular channel
    switch (aFTMChannel->timerFunction)
    {
      case TIMER_FUNCTION_OUTPUT_COMPARE:
        FTM0_CnSC(aFTMChannel->channelNb) = ((aFTMChannel->ioType.outputAction
            << FTM_CnSC_ELSA_SHIFT) |
                                            (aFTMChannel->timerFunction <<
                                                FTM_CnSC_MSA_SHIFT));
        break;
      case TIMER_FUNCTION_INPUT_CAPTURE:
        FTM0_CnSC(aFTMChannel->channelNb) = ((aFTMChannel->ioType.inputDetection
             << FTM_CnSC_ELSA_SHIFT) |
                                            (aFTMChannel->timerFunction <<
                                                FTM_CnSC_MSA_SHIFT));
        break;
    }

    //Initialize the ChannelCallBack
    FTM0Channels[aFTMChannel->channelNb].timerFunction = aFTMChannel->
        timerFunction;
    FTM0Channels[aFTMChannel->channelNb].userFunction  = aFTMChannel->
        userFunction;
    FTM0Channels[aFTMChannel->channelNb].userArguments = aFTMChannel->
        userArguments;
```

```
    return(bTRUE);
  }

  return (bFALSE);
}

void FTM_StartTimer(const TFTMChannel* const aFTMChannel)
{
  switch (aFTMChannel->timerFunction)
  {
    case TIMER_FUNCTION_OUTPUT_COMPARE:
      FTM0_CnV(aFTMChannel->channelNb) = FTM0_CNT + aFTMChannel->delayCount;

      if (FTM0_CnSC(aFTMChannel->channelNb) & FTM_CnSC_CHF_MASK)
      {
        FTM0_CnSC(aFTMChannel->channelNb) &= ~FTM_CnSC_CHF_MASK;
      }

      break;
    case TIMER_FUNCTION_INPUT_CAPTURE:
      //Yet to Implement;
      //....
      break;
  }

  FTM0_CnSC(aFTMChannel->channelNb) |= FTM_CnSC_CHIE_MASK; //Enable Interrupts
}

void __attribute__ ((interrupt)) FTM0_ISR(void)
{
  int i;

  for (i = 0; i < 8; i++)
  {
    if ((FTM0_CnSC(i) & FTM_CnSC_CHF_MASK) && (FTM0_CnSC(i) & FTM_CnSC_CHIE_MASK
        ))
    {
      if (FTM0Channels[i].timerFunction == TIMER_FUNCTION_OUTPUT_COMPARE)
      {
        FTM0_CnSC(i) &= ~FTM_CnSC_CHF_MASK;   //Clear the Interrupt Flag
        FTM0_CnSC(i) &= ~FTM_CnSC_CHIE_MASK;  //Stop any future interrupts from
            occurring (stop the timer)

        OS_ISREnter();
        FTM0Channels[i].userFunction(FTM0Channels[i].userArguments);  //Invoke
            the CallBackFunction
        OS_ISRExit();
      }
    }
  }
}
/* END FTM */
/*!
** @}
*/
```

# 9 FTM.h

```
/*! @file
 *
 *  @brief Routines for setting up the FlexTimer module (FTM) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the FlexTimer module (FTM).
 *
 *  @author PMcL
 *  @date 2015-09-04
 */

#ifndef FTM_H
#define FTM_H

// new types
#include "types.h"

typedef enum
{
  TIMER_FUNCTION_INPUT_CAPTURE,
  TIMER_FUNCTION_OUTPUT_COMPARE
} TTimerFunction;

typedef enum
{
  TIMER_OUTPUT_DISCONNECT,
  TIMER_OUTPUT_TOGGLE,
  TIMER_OUTPUT_LOW,
  TIMER_OUTPUT_HIGH
} TTimerOutputAction;

typedef enum
{
  TIMER_INPUT_OFF,
  TIMER_INPUT_RISING,
  TIMER_INPUT_FALLING,
  TIMER_INPUT_ANY
} TTimerInputDetection;

typedef struct
{
  uint8_t channelNb;
  uint16_t delayCount;
  TTimerFunction timerFunction;
  union
  {
    TTimerOutputAction outputAction;
    TTimerInputDetection inputDetection;
  } ioType;
  void (*userFunction)(void*);
  void *userArguments;
} TFTMChannel;


/*! @brief Sets up the FTM before first use.
```

```
 *
 *  Enables the FTM as a free running 16-bit counter.
 *  @return BOOL - TRUE if the FTM was successfully initialized.
 */
BOOL FTM_Init();

/*! @brief Sets up a timer channel.
 *
 *  @param aFTMChannel is a structure containing the parameters to be used in
     setting up the timer channel.
 *    channelNb is the channel number of the FTM to use.
 *    delayCount is the delay count (in module clock periods) for an output
     compare event.
 *    timerFunction is used to set the timer up as either an input capture or an
      output compare.
 *    ioType is a union that depends on the setting of the channel as input
     capture or output compare:
 *      outputAction is the action to take on a successful output compare.
 *      inputDetection is the type of input capture detection.
 *    userFunction is a pointer to a user callback function.
 *    userArguments is a pointer to the user arguments to use with the user
     callback function.
 *  @return BOOL - TRUE if the timer was set up successfully.
 *  @note Assumes the FTM has been initialized.
 */
BOOL FTM_Set(const TFTMChannel* const aFTMChannel);


/*! @brief Starts a timer if set up for output compare.
 *
 *  @param aFTMChannel is a structure containing the parameters to be used in
     setting up the timer channel.
 *  @note Assumes the FTM has been initialized.
 */
void FTM_StartTimer(const TFTMChannel* const aFTMChannel);


/*! @brief Interrupt service routine for the FTM.
 *
 *  If a timer channel was set up as output compare, then the user callback
     function will be called.
 *  @note Assumes the FTM has been initialized.
 */
void __attribute__ ((interrupt)) FTM0_ISR(void);

#endif
```

# 10 Flash.c

```
/*! @file
 *
 *  @brief Routines for erasing and writing to the Flash.
 *
 *  This contains the functions needed for accessing the internal Flash.
 *
 *  @author BAllen, APope
 *  @date 2015-08-14
 */
/*!
**  @addtogroup Flash_module Flash module documentation
**  @{
*/
/* MODULE FLASH */
#include "Cpu.h"
#include "MK70F12.h"
#include "Flash.h"

#define ADDRBYTEMASK       0x0000000F           //Used to get obtain the LSByte
    of an Address
#define ERASE_SECTOR_CMD   0x09                     //The 'Erase Sector' Flash
    command
#define PROGRAM_PHRASE_CMD 0x07                     //The 'Program Phrase' Flash
     command

// A structure which is used to initialize the flash command registers for a
    particular write operation.
/*!
 * @struct Flash_Command
 */
typedef struct
{
  uint8_t FCMD;                       /*!< The flash command code */
  uint8_t AddressByte2;          /*!< Flash address [23:16] */
  uint8_t AddressByte1;          /*!< Flash address [15:8] */
  uint8_t AddressByte0;          /*!< Flash address [7:0] */
  uint8_t Data[8];                  /*!< Flash Data */
} Flash_Command;

//An array of address locations that represents Flash Block 2, Sector 0, phrase
    0
unsigned char volatile * const Flash_AddrRange[8] = {
    FLASHMEMADDR0,
    FLASHMEMADDR1,
    FLASHMEMADDR2,
    FLASHMEMADDR3,
    FLASHMEMADDR4,
    FLASHMEMADDR5,
    FLASHMEMADDR6,
    FLASHMEMADDR7,
};

//An array of flash address locations available for allocation
static unsigned char volatile * FreeFlashAddr[8];
```

```
static BOOL initCommandRegisters(Flash_Command * commandRegisters, uint8_t FCMD)
    ;
static void flashRead(Flash_Command * commandRegisters);
static BOOL flashCommandExecute(Flash_Command * const commandRegisters);

/*! @brief Enables the Flash module.
 *
 *  @return BOOL - TRUE if the Flash was setup successfully.
 */
BOOL Flash_Init(void)
{
  //Enable clock gating for flash memory
  SIM_SCGC3 |= SIM_SCGC3_NFC_MASK;

  //Initialize the Flash memory allocation array
  FreeFlashAddr[0] = FLASHMEMADDR0;
  FreeFlashAddr[1] = FLASHMEMADDR1;
  FreeFlashAddr[2] = FLASHMEMADDR2;
  FreeFlashAddr[3] = FLASHMEMADDR3;
  FreeFlashAddr[4] = FLASHMEMADDR4;
  FreeFlashAddr[5] = FLASHMEMADDR5;
  FreeFlashAddr[6] = FLASHMEMADDR6;
  FreeFlashAddr[7] = FLASHMEMADDR7;

  return (bTRUE);
}

/*! @brief Allocates space for a non-volatile variable in the Flash memory.
 *
 *  @param variable is the address of a pointer to a variable that is to be
     allocated space in Flash memory.
 *  @param size The size, in bytes, of the variable that is to be allocated
     space in the Flash memory. Valid values are 1, 2 and 4.
 *  @return BOOL - TRUE if the variable was allocated space in the Flash memory.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_AllocateVar(volatile void **variable, const uint8_t size)
{
  int i;
  BOOL spaceAllocated = bFALSE;

  EnterCritical();

  /* In this code, depending on the size of the variable, we will loop through
      the AvailableFlashAddress
   * array, and check to see if that flash Address is available for assignment.
       An array element of '0'
   * indicates that that address has already been taken.
   *
   * We can safely assume that the user will never want to obtain the address '0
       x0000000', as we know it is
   * not part of our flash address range for our micro-controller.
   */
  if (size == 1 || size == 2 || size == 4)
  {
    for (i = 0; i < sizeof(FreeFlashAddr); i+=size)
    {
```

```
      if (FreeFlashAddr[i] != 0)
      {
        if (size == 1)
        {
          *variable = FreeFlashAddr[i];          //Assign the address to the
              variable
          FreeFlashAddr[i] = 0;                  //Clear this element to indicate
               that the address is no longer available
          spaceAllocated = bTRUE;
          break;
        }
        else if (size == 2)
        {
          if (FreeFlashAddr[i+1] != 0) //Two spaces are required to store this
               data
          {
            *variable = FreeFlashAddr[i];
            FreeFlashAddr[i] = 0;
            FreeFlashAddr[i+1] = 0;
            spaceAllocated = bTRUE;
            break;
          }
        }
        else if (size == 4)
        {
          //Four spaces are required to store this data
          if (FreeFlashAddr[i+1] != 0 && FreeFlashAddr[i+2] != 0 &&
               FreeFlashAddr[i+3] != 0)
          {
            *variable = FreeFlashAddr[i];

            FreeFlashAddr[i] = 0;
            FreeFlashAddr[i+1] = 0;
            FreeFlashAddr[i+2] = 0;
            FreeFlashAddr[i+3] = 0;
            spaceAllocated = bTRUE;
            break;
          }
        }
      }
    }
  }

  ExitCritical();

  return (spaceAllocated);
}

/*! @brief Writes a 32-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 32-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if address is
      not aligned to a 4-byte boundary or if there is a programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write32(uint32_t volatile * const address, const uint32_t data)
```

```
{
  Flash_Command flashWrite;
  uint8_t allignedByte;

  //Variables to store parts of data
  uint32union_t dataLongWord;
  uint16union_t dataWord0, dataWord1;
  uint8_t byte0, byte1, byte2, byte3;

  //Splitting 'data' into easy accessible bytes
  dataLongWord.l = data;
  dataWord0.l = dataLongWord.s.Hi; dataWord1.l = dataLongWord.s.Lo;
  byte0 = dataWord0.s.Hi;
  byte1 = dataWord0.s.Lo;
  byte2 = dataWord1.s.Hi;
  byte3 = dataWord1.s.Lo;

  if (initCommandRegisters(&flashWrite, PROGRAM_PHRASE_CMD)) //Populate the
      registers with address and data information
  {
    if (((uint32_t) address % 4) == 0)
    {
      /* We want to be able to determine what byte this address is aligned upon,
           and therefore determine
       * which bytes of the phrase we are wishing to modify.
       */
      allignedByte = ((uint32_t) address & ADDRBYTEMASK);
      allignedByte = (allignedByte % 8);

      //Modify the data registers accordingly
      flashWrite.Data[allignedByte]     = byte0;
      flashWrite.Data[(allignedByte+1)] = byte1;
      flashWrite.Data[(allignedByte+2)] = byte2;
      flashWrite.Data[(allignedByte+3)] = byte3;

      if (Flash_Erase())
      {
        return (flashCommandExecute(&flashWrite));
      }
    }
  }

  return (bFALSE);
}

/*! @brief Writes a 16-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 16-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if address is
      not aligned to a 2-byte boundary or if there is a programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write16(uint16_t volatile * const address, const uint16_t data)
{
  Flash_Command flashWrite;
  uint8_t allignedByte;
```

```
  //Variables to store parts of data
  uint16union_t dataWord;
  uint8_t byte0, byte1;

  //Splitting 'data' into easy accessible bytes
  dataWord.l = data;
  byte0 = dataWord.s.Hi;
  byte1 = dataWord.s.Lo;

  if (initCommandRegisters(&flashWrite, PROGRAM_PHRASE_CMD)) //Populate the
      FFCOB registers with address and data information
  {
    if (((uint32_t) address % 2) == 0)
    {
      /* We want to be able to determine what byte this address is aligned upon,
           and therefore determine
       * which bytes of the phrase we are wishing to modify.
       */
      allignedByte = ((uint32_t) address & ADDRBYTEMASK);
      allignedByte = (allignedByte % 8);

      //Modify the data registers accordingly
      flashWrite.Data[allignedByte]     = byte0;
      flashWrite.Data[(allignedByte+1)] = byte1;

      if (Flash_Erase())
      {
        return (flashCommandExecute(&flashWrite));
      }
    }
  }

  return (bFALSE);
}

/*! @brief Writes an 8-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 8-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if there is a
     programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write8(uint8_t volatile * const address, const uint8_t data)
{
  Flash_Command flashWrite;
  uint8_t allignedByte;

  if (initCommandRegisters(&flashWrite, PROGRAM_PHRASE_CMD))    //Populate the
      FFCOB with address and current data information
  {
    /* We want to be able to determine what byte this address is aligned upon,
         and therefore determine
     * which bytes of the phrase we are wishing to modify.
     */
    allignedByte = ((uint32_t) address & ADDRBYTEMASK);
```

```
    allignedByte = (allignedByte % 8);

    //Modify the desired byte of the data accordingly
    flashWrite.Data[allignedByte] = data;

    if (Flash_Erase())
    {
      return (flashCommandExecute(&flashWrite));
    }
  }

  return (bFALSE);
}

/*! @brief Erases the entire Flash sector.
 *
 *  @return BOOL - TRUE if the Flash "data" sector was erased successfully.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Erase(void)
{
  Flash_Command flashErase;

  if (initCommandRegisters(&flashErase, ERASE_SECTOR_CMD)) //Populate the
      registers with address information
  {
    return (flashCommandExecute(&flashErase));
  }

  return (bFALSE);
}

/*! @brief Populates a flash command struct with address and FCMD information.
 *
 *  @param commandRegisters The Flash_Command struct to populate.
 *  @param FCMD The flash command code.
 *  @return BOOL - TRUE if Struct populated, FALSE if unrecognized FCMD.
 *  @note Assumes Flash has been initialized.
 */
static BOOL initCommandRegisters(Flash_Command * commandRegisters, uint8_t FCMD)
{
  //In our micro-controller we are limited to the flash address range 0
      x0008_0000 to 0x0008_0007
  //Therefore we know we can initialize the FFCOB registers containing address
      information to:
  commandRegisters->AddressByte2 = 0x08;
  commandRegisters->AddressByte1 = 0x00;
  commandRegisters->AddressByte0 = 0x00;

  switch (FCMD)
  {
    case ERASE_SECTOR_CMD:                              //Flash 'Erase Sector'
      commandRegisters->FCMD = FCMD;
      break;
    case PROGRAM_PHRASE_CMD:                            //Flash 'Program Phrase'
      commandRegisters->FCMD = FCMD;
      flashRead(commandRegisters); //Store the current contents of flash memory
```

```
          into the data section of the FFCOB registers
      break;
    default:
      return (bFALSE);
      break;
  }

  return (bTRUE);
}

/*! @brief Populates a flash command struct with the current contents of flash
    memory.
 *
 *  @param commandRegisters The Flash_Command struct to populate with data
     information.
 *  @return void
 */
static void flashRead(Flash_Command * commandRegisters)
{
  int i;
  for (i = 0; i < sizeof(commandRegisters); i++)
  {
    commandRegisters->Data[i] = *(Flash_AddrRange[i]);
  }
}

/*! @brief Executes a flash command operation in hardware.
 *
 *  @param commandRegisters The desired flash command to execute in hardware.
 *  @return BOOL - TRUE if operation completed successfully.
 *  @note Assumes Flash has been initialized.
 */
static BOOL flashCommandExecute(Flash_Command * const commandRegisters)
{
  while (!(FTFE_FSTAT & FTFE_FSTAT_CCIF_MASK)); //Waiting for the previous flash
       command (if any) to complete

  //Clear any erroneous error bits it may have set
  FTFE_FSTAT = FTFE_FSTAT_ACCERR_MASK | FTFE_FSTAT_FPVIOL_MASK |
      FTFE_FSTAT_RDCOLERR_MASK;

  FTFE_FCCOB0 = commandRegisters->FCMD;               //Code that defines the
      FTFE command

  FTFE_FCCOB1 = commandRegisters->AddressByte2; //FLash Address [23:16]
  FTFE_FCCOB2 = commandRegisters->AddressByte1; //FLash Address [15:8]
  FTFE_FCCOB3 = commandRegisters->AddressByte0; //FLash Address [7:0]

  FTFE_FCCOB4 = commandRegisters->Data[3];          //The data is written using
      a big endian convention
  FTFE_FCCOB5 = commandRegisters->Data[2];
  FTFE_FCCOB6 = commandRegisters->Data[1];
  FTFE_FCCOB7 = commandRegisters->Data[0];
  FTFE_FCCOB8 = commandRegisters->Data[7];
  FTFE_FCCOB9 = commandRegisters->Data[6];
  FTFE_FCCOBA = commandRegisters->Data[5];
  FTFE_FCCOBB = commandRegisters->Data[4];
```

```
  FTFE_FSTAT = FTFE_FSTAT_CCIF_MASK;                                  //Set the CCIF
      Flag indicating that we are ready to execute
  while (!(FTFE_FSTAT & FTFE_FSTAT_CCIF_MASK));                 //Wait for hardware to
      execute the command

  return (!(FTFE_FSTAT & FTFE_FSTAT_MGSTAT0_MASK));          //Return bFALSE for any
      errors it may have caused
}
/* END FLASH */
/*!
** @}
*/
```

# 11 Flash.h

```
/*! @file
 *
 *  @brief Routines for erasing and writing to the Flash.
 *
 *  This contains the functions needed for accessing the internal Flash.
 *
 *  @author PMcL
 *  @date 2015-08-14
 */

#ifndef FLASH_H
#define FLASH_H

// new types
#include "types.h"

/* The pre-defined 'available' address space in Flash for our Micro controller
    */
#define FLASHMEMADDR0 (unsigned char volatile *) (0x00080000)
#define FLASHMEMADDR1 (unsigned char volatile *) (0x00080001)
#define FLASHMEMADDR2 (unsigned char volatile *) (0x00080002)
#define FLASHMEMADDR3 (unsigned char volatile *) (0x00080003)
#define FLASHMEMADDR4 (unsigned char volatile *) (0x00080004)
#define FLASHMEMADDR5 (unsigned char volatile *) (0x00080005)
#define FLASHMEMADDR6 (unsigned char volatile *) (0x00080006)
#define FLASHMEMADDR7 (unsigned char volatile *) (0x00080007)

//An array of address locations that represents Flash Block 2, Sector 0, phrase
    0
extern unsigned char volatile * const Flash_AddrRange[8];

/*! @brief Enables the Flash module.
 *
 *  @return BOOL - TRUE if the Flash was setup successfully.
 */
BOOL Flash_Init(void);

/*! @brief Allocates space for a non-volatile variable in the Flash memory.
 *
 *  @param variable is the address of a pointer to a variable that is to be
     allocated space in Flash memory.
 *          The pointer will be allocated to a relevant address:
 *          If the variable is a byte, then any address.
 *          If the variable is a half-word, then an even address.
 *          If the variable is a word, then an address divisible by 4.
 *          This allows the resulting variable to be used with the relevant
     Flash_Write function which assumes a certain memory address.
 *          e.g. a 16-bit variable will be on an even address
 *  @param size The size, in bytes, of the variable that is to be allocated
     space in the Flash memory. Valid values are 1, 2 and 4.
 *  @return BOOL - TRUE if the variable was allocated space in the Flash memory.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_AllocateVar(volatile void **variable, const uint8_t size);
```

```
/*! @brief Writes a 32-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 32-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if address is
     not aligned to a 4-byte boundary or if there is a programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write32(uint32_t volatile * const address, const uint32_t data);

/*! @brief Writes a 16-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 16-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if address is
     not aligned to a 2-byte boundary or if there is a programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write16(uint16_t volatile * const address, const uint16_t data);

/*! @brief Writes an 8-bit number to Flash.
 *
 *  @param address The address of the data.
 *  @param data The 8-bit data to write.
 *  @return BOOL - TRUE if Flash was written successfully, FALSE if there is a
     programming error.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Write8(uint8_t volatile * const address, const uint8_t data);

/*! @brief Erases the entire Flash sector.
 *
 *  @return BOOL - TRUE if the Flash "data" sector was erased successfully.
 *  @note Assumes Flash has been initialized.
 */
BOOL Flash_Erase(void);
#endif
```

# 12 LEDs.c

```
/*! @file
 *
 *  @brief Routines to access the LEDs on the TWR-K70F120M.
 *
 *  This contains the functions for operating the LEDs.
 *
 *  @author BAllen, APope
 *  @date 2015-08-15
 */
/*!
**  @addtogroup LED_module LED module documentation
**  @{
*/
/* MODULE LED */
#include "MK70F12.h"
#include "LEDs.h"

BOOL LEDs_Init(void)
{
  SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; //Enable PORTA clock for GPIO

  //Initialize the state of the output pins (logic High is off)
  GPIOA_PSOR |= (LED_ORANGE | LED_BLUE | LED_YELLOW | LED_GREEN);

  //Set PORTA_PCR10, 11, 28, 29 to (Alternative 1): General purpose I/O Pin
  PORTA_PCR(11) &= ~PORT_PCR_MUX_MASK;
  PORTA_PCR(28) &= ~PORT_PCR_MUX_MASK;
  PORTA_PCR(29) &= ~PORT_PCR_MUX_MASK;
  PORTA_PCR(10) &= ~PORT_PCR_MUX_MASK;

  PORTA_PCR(11) |= PORT_PCR_MUX(1); //ORANGE
  PORTA_PCR(28) |= PORT_PCR_MUX(1); //YELLOW
  PORTA_PCR(29) |= PORT_PCR_MUX(1); //GREEN
  PORTA_PCR(10) |= PORT_PCR_MUX(1); //BLUE

  //Set the direction for each pin (1: output)
  GPIOA_PDDR |= (LED_ORANGE | LED_BLUE | LED_YELLOW | LED_GREEN);

  return (bTRUE);
}

void LEDs_On(const TLED colour)
{
  GPIOA_PCOR |= colour;
}

void LEDs_Off(const TLED colour)
{
  GPIOA_PSOR |= colour;
}

void LEDs_Toggle(const TLED colour)
{
  GPIOA_PTOR |= colour;
}
```

```
void LEDs_AllOff(void)
{
  LEDs_Off(LED_YELLOW);
  LEDs_Off(LED_ORANGE);
  LEDs_Off(LED_GREEN);
  LEDs_Off(LED_BLUE);
}

void LEDs_ResetToTowerInit(void)
{
  LEDs_Off(LED_YELLOW);
  LEDs_On(LED_ORANGE);          //Orange LED indicates successful tower init
  LEDs_Off(LED_GREEN);
  LEDs_Off(LED_BLUE);
}

/* END LED */
/*!
** @}
*/
```

# 13 LEDs.h

```
/*! @file
 *
 *  @brief Routines to access the LEDs on the TWR-K70F120M.
 *
 *  This contains the functions for operating the LEDs.
 *
 *  @author PMcL
 *  @date 2015-08-15
 */

#ifndef LEDS_H
#define LEDS_H

// new types
#include "types.h"

typedef enum
{
  LED_ORANGE = (1 << 11),
  LED_YELLOW = (1 << 28),
  LED_GREEN  = (1 << 29),
  LED_BLUE   = (1 << 10)
} TLED;

/*! @brief Sets up the LEDs before first use.
 *
 *  @return BOOL - TRUE if the LEDs were successfully initialized.
 */
BOOL LEDs_Init(void);

/*! @brief Turns an LED on.
 *
 *  @param colour The colour of the LED to turn on.
 *  @note Assumes that LEDs_Init has been called.
 */
void LEDs_On(const TLED colour);

/*! @brief Turns off an LED.
 *
 *  @param colour THe colour of the LED to turn off.
 *  @note Assumes that LEDs_Init has been called.
 */
void LEDs_Off(const TLED colour);

/*! @brief Toggles an LED.
 *
 *  @param colour THe colour of the LED to toggle.
 *  @note Assumes that LEDs_Init has been called.
 */
void LEDs_Toggle(const TLED colour);

/*! @brief Turns all the LEDs off
 *
 *  @note Assumes LEDs_Init has been called.
 */
```

```
void LEDs_AllOff(void);

/*! @brief Resets all the LEDs to their respective states just after a
    sucessfull Tower init.
 *
 *  @note Assumes that LEDs_Init has been called.
 */
void LEDs_ResetToTowerInit(void);
#endif
```

# 14 PIT.c

```
/*! @file
 *
 *  @brief Routines for controlling Periodic Interrupt Timer (PIT) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the periodic interrupt timer (PIT)
     .
 *
 *  @author BAllen, APope
 *  @date 2015-08-22
 */
/*!
**  @addtogroup PIT_module PIT module documentation
**  @{
*/
/* MODULE PIT */
#include "PIT.h"
#include "MK70F12.h"
#include "bits.h"
#include "OS.h"

static void (*ISRCallBack)(void *);        /*!< The callback function that the
    PIT ISR will invoke */
static void *CallBackArguments;              /*!< The arguments to pass to the
    ISRCallBack function */

static uint32_t ModuleClk;                       /*!< The PIT Module Clock in Hz
    */

BOOL PIT_Init(const uint32_t moduleClk, void (*userFunction)(void *), void *
    userArguments)
{
  ISRCallBack = userFunction;
  CallBackArguments = userArguments;

  //Enable clock gating for the PIT module
  SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;

  //Set up the Module Control Register
  PIT_MCR |= PIT_MCR_MDIS_MASK;          //Disable the PIT Module initially
  PIT_MCR |= PIT_MCR_FRZ_MASK;           //Freeze timer during debug

  //Set the PIT module clock
  ModuleClk = moduleClk;

  NVICICPR2 = (1 << 4);                  //Clear pending interrupts for PIT in
      NVIC
  NVICISER2 = (1 << 4);                  //Enable interrupts from PIT module in
      NVIC

  PIT_MCR &= ~PIT_MCR_MDIS_MASK;         //Enable the PIT module
  return (bTRUE);
}

void PIT_Set(const uint32_t period, const BOOL restart)
```

```
{
  PIT_LDVAL0 = (((ModuleClk)/(1000000000/period)) - 1); //The amount of 'ticks'
      for the desired period

  if (restart)
  {
    PIT_Enable(bFALSE); //Disable the PIT Timer
    PIT_Enable(bTRUE);  //Re-enable the Timer with the new value of LDVAL
  }

  //Enable PIT Interrupts
  PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK;
}

void PIT_Enable(const BOOL enable)
{
  if (enable)
  {
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;
  }
  else
  {
    PIT_TCTRL0 &= ~PIT_TCTRL_TEN_MASK;
  }
}

void __attribute__ ((interrupt)) PIT_ISR(void)
{
  PIT_TFLG0 |= PIT_TFLG_TIF_MASK; //Clear the Interrupt Flag

  OS_ISREnter();
  ISRCallBack(CallBackArguments); //Invoke the Call-back function
  OS_ISRExit();
}
/* END PIT */
/*!
** @}
*/
```

# 15 PIT.h

```
/*! @file
 *
 *  @brief Routines for controlling Periodic Interrupt Timer (PIT) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the periodic interrupt timer (PIT)
     .
 *
 *  @author PMcL
 *  @date 2015-08-22
 */

#ifndef PIT_H
#define PIT_H

// new types
#include "types.h"

/*! @brief Sets up the PIT before first use.
 *
 *  Enables the PIT and freezes the timer when debugging.
 *  @param moduleClk The module clock rate in Hz.
 *  @param userFunction is a pointer to a user callback function.
 *  @param userArguments is a pointer to the user arguments to use with the user
       callback function.
 *  @return BOOL - TRUE if the PIT was successfully initialized.
 *  @note Assumes that moduleClk has a period which can be expressed as an
     integral number of nanoseconds.
 */
BOOL PIT_Init(const uint32_t moduleClk, void (*userFunction)(void *), void *
    userArguments);

/*! @brief Sets the value of the desired period of the PIT.
 *
 *  @param period The desired value of the timer period in nanoseconds.
 *  @param restart TRUE if the PIT is disabled, a new value set, and then
     enabled.
 *                 FALSE if the PIT will use the new value after a trigger event
     .
 *  @note The function will enable the timer and interrupts for the PIT.
 */
void PIT_Set(const uint32_t period, const BOOL restart);

/*! @brief Enables or disables the PIT.
 *
 *  @param enable - TRUE if the PIT is to be enabled, FALSE if the PIT is to be
     disabled.
 */
void PIT_Enable(const BOOL enable);

/*! @brief Interrupt service routine for the PIT.
 *
 *  The periodic interrupt timer has timed out.
 *  The user callback function will be called.
 *  @note Assumes the PIT has been initialized.
```

```
 */
void __attribute__ ((interrupt)) PIT_ISR(void);

#endif
```

# 16 RTC.c

```
/*! @file
 *
 *  @brief Routines for controlling the Real Time Clock (RTC) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the real time clock (RTC).
 *
 *  @author BAllen, APope
 *  @date 2015-08-24
 */
/*!
**  @addtogroup RTC_module RTC module documentation
**  @{
*/
/* MODULE RTC */
#include "MK70F12.h"
#include "RTC.h"
#include "OS.h"

static void (*ISRCallBack)(void *);        /*!< The callback function that the
    RTC ISR will invoke */
static void *CallBackArguments;              /*!< The arguments to pass to the
    ISRCallBack function */

BOOL RTC_Init(void (*userFunction)(void *), void *userArguments)
{
  int i;
  uint8_t hours, seconds, minutes;
  uint32_t totalSeconds;

  //Assign the Callback
  ISRCallBack = userFunction;
  CallBackArguments = userArguments;

  //Enable Clock gating
  SIM_SCGC6 |= SIM_SCGC6_RTC_MASK;

  //Attempt to assert SWR flag to check if the control register is locked
  RTC_CR = RTC_CR_SWR_MASK;

  if (RTC_CR == RTC_CR_SWR_MASK)
  {
    RTC_CR &= ~RTC_CR_SWR_MASK; //Software Reset: 0 No effect

    RTC_TSR = 0;                //Clear the TIF flag

    //Setup the RTC control register
    RTC_CR |= RTC_CR_SC2P_MASK;   //Oscillator 2pF Load Configure: 1 enabled
    RTC_CR &= ~RTC_CR_SC4P_MASK;  //Oscillator 4pF Load Configure: 0 disabled
    RTC_CR &= ~RTC_CR_SC8P_MASK;  //Oscillator 8pF Load Configure: 0 disabled
    RTC_CR |= RTC_CR_SC16P_MASK;  //Oscillator 16pF Load Configure: 1 enabled

    RTC_CR &= ~RTC_CR_CLKO_MASK;  //Clock Output: 0 The 32 kHz clock is output
        to no other peripherals
    RTC_CR |= RTC_CR_OSCE_MASK;   //Oscillator Enable: 1 32.768 kHz oscillator
```

```
        is enabled

    RTC_CR &= ~RTC_CR_UM_MASK;     //Update Mode: 0 Registers cannot be written
        when locked
    RTC_CR &= ~RTC_CR_SUP_MASK;    //Supervisor Access: 0 Non-supervisor mode
        write accesses are not supported

    RTC_CR &= ~RTC_CR_WPE_MASK;    //Wakeup Pin Enable: 0 Wakeup pin is disabled

    //Wait an arbitrary amount of time for the oscillator to become stable
        before enabling the time counter
    for(i = 0; i < 9000; i++);

    RTC_LR &= ~RTC_LR_CRL_MASK; //Control Register Lock: 0 is locked and writes
        are ignored
  }

  //Setup the RTC_LR (Lock Register)
  RTC_LR &= ~RTC_LR_MCHL_MASK;  //Monotonic Counter High Lock: 0 is locked and
      writes are ignored
  RTC_LR &= ~RTC_LR_MCLL_MASK;  //Monotonic Counter Low Lock: 0 is locked and
      writes are ignored
  RTC_LR &= ~RTC_LR_MEL_MASK;   //Monotonic Enable Lock: 0 is locked and writes
      are ignored
  RTC_LR &= ~RTC_LR_TTSL_MASK;  //Tamper Time seconds Lock: 0 is locked and
      writes are ignored
  RTC_LR &= ~RTC_LR_LRL_MASK;   //Lock Register Lock: 0 is locked and writes are
       ignored
  RTC_LR |= RTC_LR_SRL_MASK;    //Status Register Lock: 1 writes complete as
      normal
  RTC_LR &= ~RTC_LR_TCL_MASK;   //Time compensation Lock: 0 is locked and writes
       are ignored

  //Init NVIC
  NVICICPR2 = (1 << 3); //Clear pending interrupts on RTC
  NVICISER2 = (1 << 3); //Enable interrupts from RTC module

  //Enable Time Seconds Interrupt and Disable the others
  RTC_IER |= RTC_IER_TSIE_MASK;
  RTC_IER &= ~RTC_IER_TAIE_MASK;
  RTC_IER &= ~RTC_IER_TOIE_MASK;
  RTC_IER &= ~RTC_IER_TIIE_MASK;

  RTC_SR |= RTC_SR_TCE_MASK;
  return (bTRUE);
}

void RTC_Set(const uint8_t hours, const uint8_t minutes, const uint8_t seconds)
{
  uint32_t totalSeconds = seconds + (minutes * 60) + (hours * 60 * 60);

  //Disable the Time counter
  RTC_SR &= ~RTC_SR_TCE_MASK;

  //Write the new time value to RTC_TSR
  RTC_TSR = totalSeconds;
```

```
  //Re-enable the Time Counter
  RTC_SR |= RTC_SR_TCE_MASK;
}

void RTC_Get(uint8_t * const hours, uint8_t * const minutes, uint8_t * const
    seconds)
{
  uint32_t Read1, Read2;
  uint8_t days;
  BOOL timeMatch = bFALSE;

  /* Reading the timer counter while incrementing may return invalid data due to
       sync issues. We must perform
   * two read accesses and confirm that the same data was returned for both
       reads.
   */
  while (!timeMatch)
  {
    Read1 = RTC_TSR; Read2 = RTC_TSR;
    timeMatch = (Read1 == Read2);
  }

  RTC_CalcTime(Read1, &days, hours, minutes, seconds);
}

void RTC_CalcTime(uint32_t totalSeconds, uint8_t * const days, uint8_t * const
    hours, uint8_t * const minutes, uint8_t *const seconds)
{
  *days    = ((totalSeconds / 86400));
  *hours   = ((totalSeconds / 3600) % 24);
  *minutes = ((totalSeconds % 3600) / 60);
  *seconds = ((totalSeconds % 3600) % 60);
}

void __attribute__ ((interrupt)) RTC_ISR(void)
{
  OS_ISREnter();
  ISRCallBack(CallBackArguments);       //Invoke the ISR Callback function
  OS_ISRExit();
}
/* END RTC */
/*!
** @}
*/
```

# 17 RTC.h

```
/*! @file
 *
 *  @brief Routines for controlling the Real Time Clock (RTC) on the TWR-
     K70F120M.
 *
 *  This contains the functions for operating the real time clock (RTC).
 *
 *  @author PMcL
 *  @date 2015-08-24
 */

#ifndef RTC_H
#define RTC_H

// new types
#include "types.h"

/*! @brief Initializes the RTC before first use.
 *
 *  Sets up the control register for the RTC and locks it.
 *  Enables the RTC and sets an interrupt every second.
 *  @param userFunction is a pointer to a user callback function.
 *  @param userArguments is a pointer to the user arguments to use with the user
      callback function.
 *  @return BOOL - TRUE if the RTC was successfully initialized.
 */
BOOL RTC_Init(void (*userFunction)(void *), void *userArguments);

/*! @brief Sets the value of the real time clock.
 *
 *  @param hours The desired value of the real time clock hours (0-23).
 *  @param minutes The desired value of the real time clock minutes (0-59).
 *  @param seconds The desired value of the real time clock seconds (0-59).
 *  @note Assumes that the RTC module has been initialized and all input
     parameters are in range.
 */
void RTC_Set(const uint8_t hours, const uint8_t minutes, const uint8_t seconds);

/*! @brief Gets the value of the real time clock.
 *
 *  @param hours The address of a variable to store the real time clock hours.
 *  @param minutes The address of a variable to store the real time clock
     minutes.
 *  @param seconds The address of a variable to store the real time clock
     seconds.
 *  @note Assumes that the RTC module has been initialized.
 */
void RTC_Get(uint8_t* const hours, uint8_t* const minutes, uint8_t* const
    seconds);

/*! @brief Gets the value of the real time clock.
 *
 *  @param totalSeconds The total clock seconds that have elapsed.
 *  @param days         The address of a variable to store the days.
 *  @param hours        The address of a variable to store the clock hours.
```

```
 *  @param minutes      The address of a variable to store the clock minutes.
 *  @param seconds      The address of a variable to store the clock seconds.
 */
void RTC_CalcTime(uint32_t totalSeconds, uint8_t * const days, uint8_t * const
    hours, uint8_t * const minutes, uint8_t *const seconds);

/*! @brief Interrupt service routine for the RTC.
 *
 *  The RTC has incremented one second.
 *  The global variable RTC_Updated will be set to bTRUE.
 *  @note Assumes the RTC has been initialized.
 */
void __attribute__ ((interrupt)) RTC_ISR(void);

#endif
```

# 18 UART.c

```
/*! @file
 *
 *  @brief I/O routines for UART communications on the TWR-K70F120M.
 *
 *  This contains the functions for operating the UART (serial port).
 *
 *  @author BAllen, APope
 *  @date 2015-08-15
 */
/*!
**  @addtogroup UART_module UART module documentation
**  @{
*/
/* MODULE UART */
#include "MK70F12.h"
#include "Cpu.h"
#include "bits.h"
#include "FIFO.h"
#include "UART.h"
#include "OS.h"

static TFIFO TxFIFO;              /*!< Transmitter FIFO buffer */
static TFIFO RxFIFO;              /*!< Receiver FIFO buffer */
static OS_ECB *BufferAvailable;   /*!< Semaphore to indicate when OutChar is
    available */

BOOL UART_PrintDriver(uint8_t const * const buffer, uint8_t length)
{
  uint8_t i;
  BOOL sendOk = bTRUE;

  OS_SemaphoreWait(BufferAvailable, 0);
  for (i = 0; i < length; i++)
  {
    sendOk &= UART_OutChar(buffer[i]);
  }
  OS_SemaphoreSignal(BufferAvailable);

  return (sendOk);
}

BOOL UART_Init(const uint32_t baudRate, const uint32_t moduleClk)
{
  uint8_t brfa;             //Baud Rate Fractional Adjuster
  uint16union_t sbr;
  BOOL initOk = bFALSE;

  BufferAvailable = OS_SemaphoreCreate(1);    //Initialise the semaphore

  if (baudRate != 0) //Sanity check, Cannot Divide by zero in the baud rate
      equation below
  {
    //Integer division will automatically truncate anything after the decimal
        point, therefore we can simply find the SBR:
    sbr.l = (moduleClk / (16 * baudRate));
```

```
if (sbr.l < 0x1FFF) //Register size to hold SBR is 13 bits long only,
    therefore it cannot be greater than 0x1FFF
{
  //As we are not utilizing the FPU yet, we shall scale by a factor of 32 to
       avoid floating point numbers
  brfa = ((moduleClk*2) / (baudRate)) - ((sbr.l * 32));

  //Enable UART2 and PORTE through System Clock Gating Registers
  SIM_SCGC4 |= SIM_SCGC4_UART2_MASK;
  SIM_SCGC5 |= SIM_SCGC5_PORTE_MASK;

  //Set both PORTE_PCR16/17 to (Alternative 3): UART2_TX/UART2_RX
  PORTE_PCR16 &= ~PORT_PCR_MUX_MASK;
  PORTE_PCR17 &= ~PORT_PCR_MUX_MASK;

  // Set MUX to 3
  PORTE_PCR16 |= PORT_PCR_MUX(3);
  PORTE_PCR17 |= PORT_PCR_MUX(3);

  //Set SBR[12:8] in the UART2_BDH Register
  UART2_BDH |= UART_BDH_SBR_MASK;
  UART2_BDH &= (sbr.s.Hi |= ~(UART_BDH_SBR_MASK));  //Put the higher 5 bits
      of SBR into the lower 5 bits of the UART2_BDH register

  //Set SBR[7:0] in the UART2_BDL Register
  UART2_BDL |= 0xFF;
  UART2_BDL &= sbr.s.Lo;                            //Put the remainder 8
      bits of SBR into the UART2_BDL register

  //Set BRFA in the UART2_C4 Register
  UART2_C4 |= UART_C4_BRFA_MASK;
  UART2_C4 &= (brfa |= ~(UART_C4_BRFA_MASK));       //Put the lower 5 bits
      of BRFA into the lower 5 bits of the UART2_C4 register

  //Set Control Register 1
  UART2_C1 &= ~UART_C1_LOOPS_MASK;          //LOOPS; 0:Normal operation
  UART2_C1 &= ~UART_C1_UARTSWAI_MASK;       //UARTSWAI; 0:UART clock
      continues to run in Wait mode
  UART2_C1 &= ~UART_C1_RSRC_MASK;           //RSRC; 0:Selects internal loop
      back mode. Receiver input is internally connected to transmitter output
  UART2_C1 &= ~UART_C1_M_MASK;              //M; 0:Normal-start+ 8 data bits
       + stop
  UART2_C1 &= ~UART_C1_WAKE_MASK;           //WAKE; 0:Idle line wakeup
  UART2_C1 &= ~UART_C1_ILT_MASK;            //ILT; 0:Idle character bit
      count starts after start bit
  UART2_C1 &= ~UART_C1_PE_MASK;             //PE; 0:Parity function disabled
  UART2_C1 &= ~UART_C1_PT_MASK;             //PT; 0:Even Parity

  //Set Control Register 2
  UART2_C2 &= ~UART_C2_TIE_MASK;            //TIE; 0:TDRE Interrupt and DMA
      transfer requests disabled (initially upon startup)
  UART2_C2 &= ~UART_C2_TCIE_MASK;           //TCIE; 0:TC interrupt requests
      disabled
  UART2_C2 |= UART_C2_RIE_MASK;             //RIE; 1:RDRF interrupt and DMA
      transfer request enabled
  UART2_C2 &= ~UART_C2_ILIE_MASK;           //ILIE; 0:IDLE interrupt
```

```
        requests disabled
        UART2_C2 |= UART_C2_TE_MASK;              //TE; 1:Transmitter on
        UART2_C2 |= UART_C2_RE_MASK;              //RE; 1:Receiver on
        UART2_C2 &= ~UART_C2_RWU_MASK;            //RWU; 0:Normal operation
        UART2_C2 &= ~UART_C2_SBK_MASK ;           //SBK; 0:Normal transmitter
            operation

        //Set Control Register 3
        UART2_C3 &= ~UART_C3_TXDIR_MASK;          //TXDIR; 0:TXD pin is an input
            in single wire mode
        UART2_C3 &= ~UART_C3_TXINV_MASK;          //TXINV; 0:Transmit data is not
            inverted
        UART2_C3 &= ~UART_C3_ORIE_MASK;           //ORIE; 0:OR interrupts are
            disabled
        UART2_C3 &= ~UART_C3_NEIE_MASK;           //NEIE; 0:NF interrupt requests
            are disabled
        UART2_C3 &= ~UART_C3_FEIE_MASK;           //FEIE; 0:FE interrupt requests
            are disabled
        UART2_C3 &= ~UART_C3_PEIE_MASK;           //PEIE; 0:PF interrupt requests
            are disabled

        //Set Control Register 4
        UART2_C4 &= ~UART_C4_MAEN1_MASK;          //MAEN1; 0:All data received is
            transferred to the data buffer if MAEN2 is cleared
        UART2_C4 &= ~UART_C4_MAEN2_MASK;          //MAEN2; 0:All data received is
            transferred to the data buffer if MAEN1 is cleared

        //Set Control Register 5
        UART2_C5 &= ~UART_C5_TDMAS_MASK;          //TDMAS; 0:If C2[TIE] & S1[TDRE]
             are not set, TDRE interrupt request signal can assert
        UART2_C5 &= ~UART_C5_RDMAS_MASK;          //RDMAS; 0:If C2[RIE] & S1[RDRF]
             are not set, RDRF interrupt request signal can assert

        //Set MODEM Register
        UART2_MODEM &= ~UART_MODEM_RXRTSE_MASK;   //RXRTSE; 0:The receiver has no
            effect on RTS
        UART2_MODEM &= ~UART_MODEM_TXRTSPOL_MASK; //TXRTSPOL; 0:Transmitter RTS is
             active low
        UART2_MODEM &= ~UART_MODEM_TXRTSE_MASK;   //TXRTSE; 0:The transmitter has
            no effect on RTS
        UART2_MODEM &= ~UART_MODEM_TXCTSE_MASK;   //TXCTSE; 0:CTS has no effect on
             the transmitter

        //Initialize TxFIFO and RxFIFO
        FIFO_Init(&TxFIFO);
        FIFO_Init(&RxFIFO);

        //Initialize NVIC for the UART2 module
        NVICICPR1 = (1 << 17); //Clear pending interrupts on UART2
        NVICISER1 = (1 << 17); //Enable interrupts from UART2 module

        initOk = bTRUE;
      }
    }

    return (initOk);
}
```

```
BOOL UART_InChar(uint8_t * const dataPtr)
{
  return (FIFO_Get(&RxFIFO, dataPtr));
}

BOOL UART_OutChar(const uint8_t data)
{
  OS_DisableInterrupts();                              //A critical section
      occurs here

  if (FIFO_Put(&TxFIFO, data))
  {
    UART2_C2 |= UART_C2_TIE_MASK;        //ARM OUTPUT

    OS_EnableInterrupts();
    return (bTRUE);
  }

  OS_EnableInterrupts();
  return (bFALSE);
}

void __attribute__ ((interrupt)) UART_ISR(void)
{
  OS_ISREnter();

  if ((UART2_S1 & UART_S1_TDRE_MASK) && (UART2_C2 & UART_C2_TIE_MASK))  //Check
      if the TDRE and TIE flag is set
  {
    if (!FIFO_Get(&TxFIFO, (uint8_t * const)&UART2_D))
      UART2_C2 &= ~UART_C2_TIE_MASK;     //DISARM OUTPUT
  }

  if ((UART2_S1 & UART_S1_RDRF_MASK) && (UART2_C2 & UART_C2_RIE_MASK))  //Check
      if the RDRF and RIE flag is set
  {
    FIFO_Put(&RxFIFO, UART2_D);
  }

  OS_ISRExit();
}
/* END UART */
/*!
** @}
*/
```

# 19 UART.h

```
/*! @file
 *
 *  @brief I/O routines for UART communications on the TWR-K70F120M.
 *
 *  This contains the functions for operating the UART (serial port).
 *
 *  @author PMcL
 *  @date 2015-07-23
 */

#ifndef UART_H
#define UART_H

// new types
#include "types.h"

/*! @brief Sets up the UART interface before first use.
 *
 *  @param baudRate The desired baud rate in bits/sec.
 *  @param moduleClk The module clock rate in Hz.
 *  @return BOOL - TRUE if the UART was successfully initialized.
 */
BOOL UART_Init(const uint32_t baudRate, const uint32_t moduleClk);

/*! @brief Get a character from the receive FIFO if it is not empty.
 *
 *  @param dataPtr A pointer to memory to store the retrieved byte.
 *  @return BOOL - TRUE if the receive FIFO returned a character.
 *  @note Assumes that UART_Init has been called.
 */
BOOL UART_InChar(uint8_t* const dataPtr);

/*! @brief Put a byte in the transmit FIFO if it is not full.
 *
 *  @param data The byte to be placed in the transmit FIFO.
 *  @return BOOL - TRUE if the data was placed in the transmit FIFO.
 *  @note Assumes that UART_Init has been called.
 */
BOOL UART_OutChar(const uint8_t data);

/*! @brief Poll the UART status register to try and receive and/or transmit one
    character.
 *
 *  @return void
 *  @note Assumes that UART_Init has been called.
 */
void UART_Poll(void);

/*! @brief Print Driver for UART_OutChar
 *
 *  @param buffer - An array of chars to be sent to the PC.
 *  @param length - The length of the buffer.
 *  @return BOOL  - TRUE if all sent successfully.
 *  @note Assumes that UART_Init has been called.
 */
```

```
BOOL UART_PrintDriver(uint8_t const * const buffer, uint8_t length);

/*! @brief Interrupt service routine for the UART.
 *
 *  @note Assumes the transmit and receive FIFOs have been initialized.
 */
void __attribute__ ((interrupt)) UART_ISR(void);
#endif
```

# 20 bits.h

```
/*! @file
 *
 *  @brief Macros for bit manipulation.
 *
 *  This header file contains some helpful macros for bit manipulation of
     variables.
 *
 *  @author APope
 *  @date 2015-08-20
 */

#ifndef BITS_H
#define BITS_H

#define SET_BIT(x) 1 << (x)                                 //Create a variable of
    bit length (x+1) and set the MSB to a 1, with the rest as zeros
                                                            //Example use, setting
                                                                bit 7 in temp; temp
                                                                |= SET_BIT(7)
#define CLEAR_BIT(x) (~(1 << (x)))                          //Create a variable of
    bit length (x+1) and set the MSB to a 0, with the rest as ones
                                                            //Example use, clearing
                                                                bit 7 in temp; temp
                                                                &= CLEAR_BIT(7)
#define CHECK_BIT_SET(var, pos) ((var) & (1 <<(pos)))   //Will evaluate to 1 if
    bit (pos) is set in (var)

#endif
```

# 21 debounce.c

```
/*! @file
 *
 *  @brief Routines for setting up the general purpose debounce module.
 *
 *  This contains functions for debouncing switches, pushbuttons, touch
     sensitive interfaces etc.
 *
 *  @author BAllen, APope
 *  @date 2015-09-04
 */
/*!
**  @addtogroup DEBOUNCE_module DEBOUNCE module documentation
**  @{
*/
/* MODULE DEBOUNCE */
#include "Cpu.h"
#include "MK70F12.h"
#include "debounce.h"
#include "FTM.h"

//FTM call-back
static void debounceDelayCompleteCallBack(void * nothing);

//Timer used measure 10ms of debounce
static const TFTMChannel DEBOUNCE_TIMER = {
    .channelNb                          = 1,
    .delayCount                         = 244, //244 clock ticks: (10ms
        /(1/24414)) = 244.14
    .timerFunction                    = TIMER_FUNCTION_OUTPUT_COMPARE,
    .ioType.outputAction            = TIMER_OUTPUT_LOW,
    .ioType.inputDetection        = TIMER_INPUT_OFF,
    .userFunction                       = debounceDelayCompleteCallBack,
    .userArguments                      = 0
};

static void (*ButtonPressedFunction)(void *);
static void *ButtonPressedArguments;

/*! @brief Call-back after switch has reached debounce delay.
 *
 * When a delay of 10ms has been achieved this routine will be called and
     debouncing for a switch has completed.
 */
void debounceDelayCompleteCallBack(void * nothing)
{
  TButtonState buttonState = (GPIOD_PDIR & 0x00000001);  //PORT D, Pin 0
  PORTD_PCR0 |= PORT_PCR_IRQC(10);                       //Enable interrupts to
      occur again on this switch

  if (buttonState == LOGIC_0)
  {
    //The button was pressed - invoke the call-back
    ButtonPressedFunction(ButtonPressedArguments);
  }
}
```

```
void Debounce_Start(void)
{
  FTM_StartTimer(&DEBOUNCE_TIMER);  //Start the 10ms debounce timer
}

BOOL Debounce_Init(void (*userFunction)(void *), void *userArguments)
{
  ButtonPressedFunction   = userFunction;
  ButtonPressedArguments  = userArguments;

  return (FTM_Set(&DEBOUNCE_TIMER));
}
/* END DEBOUNCE */
/*!
** @}
*/
```

## 22 debounce.h

```
/*! @file
 *
 *  @brief Routines for setting up the general purpose debounce module.
 *
 *  This contains functions for debouncing switches, pushbuttons, touch
     sensitive interfaces etc.
 *
 *  @author BAllen, APope
 *  @date 2015-09-04
 */

#ifndef DEBOUNCE_H
#define DEBOUNCE_H

// new types
#include "types.h"

typedef enum
{
  BUTTON_SW1,
  BUTTON_SW2,
} TDebounceID;

typedef struct
{
  TDebounceID buttonID;
  void (*debounceCompleteCallbackFunction)(void *);
  void* debounceCompleteCallbackArguments;
} TDebounce;

/*! @brief Sets up the debounce module before first use.
 *
 *  Enables FTM timers for the button debouncing.
 *
 *  @param userFunction - Is invoked after debounce and registering that button
     has been pressed.
 *  @param userArguments - UserFunction arguments
 *
 *  @return BOOL - TRUE if the debounce module was successfully initialized.
 */
BOOL Debounce_Init(void (*userFunction)(void *), void *userArguments);

/*! @brief Start debouncing switch 1.
 *
 *  Begins a 10ms delay to debounce switch 1.
 *
 *  @note Assumes Debounce_Init has been called
 */
void Debounce_Start(void);
#endif
```

# 23 displayprint.c

```
/*! @file
 *
 *  @brief Prints Meter statistics to the display.
 *
 *  This module contains routines that allows a caller print meter useage stats
     to the display on the PC interface.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup DISPLAY_PRINT_module Display Print module documentation
**  @{
*/
/* MODULE DISPLAY_PRINT */
#include "Cpu.h"
#include "displayprint.h"
#include "UART.h"
#include "OS.h"
#include <stdio.h>
#include <string.h>

void DisplayPrint_Time(uint8_t days, uint8_t hours, uint8_t minutes, uint8_t
    seconds)
{
  char buffer[14];

  if (days <= 99)
  {
    snprintf(buffer, 13, "%d:%d:%d:%d\n", days, hours, minutes, seconds);
  }
  else
  {
    snprintf(buffer, 13, "xx:xx:xx:xx\n");
  }

  UART_PrintDriver(buffer, strlen(buffer));
}

void DisplayPrint_Power(uint16_t whole, uint16_t fraction)
{
  char buffer[13];

  if (whole <= 999)
  {
    snprintf(buffer, 12, "%d.%03d␣kW\n", whole, fraction);
  }
  else
  {
    snprintf(buffer, 12, "xxx.xxx␣kW\n");
  }

  UART_PrintDriver(buffer, strlen(buffer));
}
```

```
void DisplayPrint_Energy(uint16_t whole, uint16_t fraction)
{
  char buffer[14];

  if (whole <= 999)
  {
    snprintf(buffer, 13, "%i.%03d␣kWh\n", whole, fraction);
  }
  else
  {
    snprintf(buffer, 13, "xxx.xxx␣kWh\n");
  }

  UART_PrintDriver(buffer, strlen(buffer));
}

void DisplayPrint_Cost(uint16_t dollars, uint8_t cents)
{
  char buffer[10];

  if (dollars <= 9999)
  {
    snprintf(buffer, 9, "$%d.%02d\n", dollars, cents);
  }
  else
  {
    snprintf(buffer, 9, "$xxxx.xx\n");
  }

  UART_PrintDriver(buffer, strlen(buffer));
}
/* END DISPLAY_PRINT */
/*!
** @}
*/
```

# 24 displayprint.h

```
/*! @file
 *
 *  @brief Prints Meter statistics to the display.
 *
 *  This module contains routines that allows a caller print meter usage stats
     to the display on the PC interface.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef DISPLAY_PRINT_H
#define DISPLAY_PRINT_H

#include "types.h"

/*! @brief Prints Time to the PC interface
 *
 *  @param days     - Amount of days that have elapsed (0 - 99)
 *  @param hours    - Amount of hours that have elapsed in that day (0 - 23)
 *  @param minutes  - Amount of minutes that have elapsed in that day (0 - 60)
 *  @param seconds  - Amount of seconds that have elapsed in that day (0 - 60)
 *  @return void
 */
void DisplayPrint_Time(uint8_t days, uint8_t hours, uint8_t minutes, uint8_t
    seconds);

/*! @brief Prints Power to the PC interface (kW)
 *
 *  @param whole    - whole part PPP
 *  @param fraction - fractional part ppp
 *  @return void
 */
void DisplayPrint_Power(uint16_t whole, uint16_t fraction);

/*! @brief Prints Energy consumption to the PC interface (kWh)
 *
 *  @param whole    - whole part PPP
 *  @param fraction - fractional part ppp
 *  @return void
 */
void DisplayPrint_Energy(uint16_t whole, uint16_t fraction);

/*! @brief Prints running cost to the PC interface ($$$$.cc)
 *
 *  @param dollars - Dollars
 *  @param cents   - Cents
 *  @return void
 */
void DisplayPrint_Cost(uint16_t dollars, uint8_t cents);
#endif
```

# 25 interface.c

```
/*! @file
 *
 *  @brief Acts as a human machine interface between the DEM and PC.
 *
 *  This module contains the routines that process user requests.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup INTERFACE_module INTERFACE module documentation
**  @{
*/
/* MODULE INTERFACE */
#include "Cpu.h"
#include "bits.h"
#include "RTC.h"
#include "Flash.h"
#include "LEDs.h"
#include "FTM.h"
#include "debounce.h"
#include "packet.h"
#include "DAC.h"
#include "meter.h"
#include "interface.h"
#include "tariff.h"
#include "displayprint.h"
#include "OS.h"

#define BAUD_RATE 115200            //Baud Rate used to initialize UART2 module

//Defining the various PC to Tower packet commands
//Basic Protocol extension
#define TEST_MODE_COMMAND 0x10
#define TARIFF_COMMAND    0x11
#define TIME1_COMMAND     0x12
#define TIME2_COMMAND     0x13
#define POWER_COMMAND     0x14
#define ENERGY_COMMAND    0x15
#define COST_COMMAND      0x16

//Intermediate Protocol extension
#define FREQUENCY_COMMAND 0x17
#define VOLT_RMS_COMMAND  0x18
#define CURR_RMS_COMMAND  0x19
#define PFACTOR_COMMAND   0x1A

//Protocols for changing the self-test waveforms
#define VOLT_AMP_COMMAND  0x1B
#define CURR_AMP_COMMAND  0x1C
#define PHASE_COMMAND     0x1D

typedef struct
{
  uint32_t base;
```

```
  uint8_t n;
}TBASE;

const uint8_t PACKET_ACK_MASK = 0x80;      /*!< The helps determine whether the
    PC requires ACK upon arrival */

//Voltage Waveform
const uint16_t VOLTAGE_MAX_VALUE = 11584;  /*!< Max value of the voltage self-
    test waveform BASE OF 1000/2^15 */
const uint16_t VOLTAGE_MIN_VALUE = 9266;   /*!< Min value of the voltage self-
    test waveform BASE OF 1000/2^15 */
//Current Waveform
const uint16_t CURRENT_MAX_VALUE = 23174;  /*!< Max value of the voltage self-
    test waveform BASE OF 10/2^15 */
const uint16_t CURRENT_MIN_VALUE = 0;      /*!< Min value of the voltage self-
    test waveform BASE OF 1000/2^15 */

//Note, energy base has an additional scaling factor of 3600 - as the DEM is
    always in accelerated time mode where kWs = kWh.
const TBASE ENERGY_BASE     = { /*!< Base of Meter_TotalEnergy (kWh)
    (10*1000*(1.25mS*3600)/2^30*1000*3600) = 125 ((32Q30)*10,000). */
    .base = 125,
    .n = 30
};

const TBASE COST_TOTAL_BASE = { /*!< Base of Meter_TotalEnergyCost (10*1.25mS
    /2^30*1000) = 125 ((32Q30)*1x10^7). */
    .base = 125,
    .n = 30
};

//FTM timer structure for the display timeout
void displayDormantCallBack(void * nothing);
static const TFTMChannel DISPLAY_DORMANT_TIMER = {
    .channelNb              = 0,
    .delayCount             = 24414, //1s
    .timerFunction          = TIMER_FUNCTION_OUTPUT_COMPARE,
    .ioType.outputAction    = TIMER_OUTPUT_LOW,
    .ioType.inputDetection  = TIMER_INPUT_OFF,
    .userFunction           = displayDormantCallBack,
    .userArguments          = 0
};

static uint8_t  DormantDisplayTimerCounter;   /*!< Counts the number of times
    the DISPLAY_DORMANT_TIMER has triggered */
static OS_ECB *Switch1Pressed;                /*!< Semaphore used to indicate
    when switch 1 has been pressed. */
static OS_ECB *PrintToTerminal;               /*!< Semaphore used to indicate
    that data needs to be printed to the terminal. */

//De-bounce structure for Switch 1
void cycleDisplayCallBack(void * nothing);
static TDebounce Switch1 = {
  .buttonID = BUTTON_SW1,
  .debounceCompleteCallbackFunction = cycleDisplayCallBack,
  .debounceCompleteCallbackArguments = 0
};
```

```
typedef enum {
  DISPLAY_DORMANT,
  DISPLAY_METER_TIME,
  DISPLAY_AVERAGE_POWER,
  DISPLAY_TOTAL_ENERGY,
  DISPLAY_TOTAL_COST
}TDisplayState;

static TDisplayState CurrentDisplay;    /*!< The current Display state of the
    terminal. */

/*! @brief Call-back for the Display dormant counter.
 *
 */
void displayDormantCallBack(void * nothing)
{
  OS_DisableInterrupts();
  if (DormantDisplayTimerCounter == 15)
  {
    //The user has not activated switch1 in 15 seconds, reset the display to the
        dormant state
    DormantDisplayTimerCounter = 1;
    CurrentDisplay = DISPLAY_DORMANT;
  }
  else
  {
    DormantDisplayTimerCounter++;
    FTM_StartTimer(&DISPLAY_DORMANT_TIMER); //Restart the timer
  }
  OS_EnableInterrupts();
}

/*! @brief Call-back for switch 1.
 *
 *  This will cycle the state machine, and change the currently displaying
     quantity.
 */
void cycleDisplayCallBack(void * nothing)
{
  OS_SemaphoreSignal(Switch1Pressed);
}

/*! @brief Using the base definition it will normalise and calculate
    AveragePower
 *
 *  @param integerPart - Stores the Integer part of Average Power
 *  @param fractionalPart - Stores the fractional part of Average Power
 *  @param convertToKiloWatts - Bool is set if user wants the answer in
     KilloWatts
 */
static void normalisePower(uint16_t * const integerPart, uint16_t * const
    fractionalPart, BOOL convertToKiloWatts)
{
  uint64_t result;

  OS_DisableInterrupts();
```

```
  uint16_t voltRMS = Meter_VoltageRMS;
  uint16_t currRMS = Meter_CurrentRMS;
  uint32_t pf = DAC_PowerFactor();
  OS_EnableInterrupts();

  result = (uint64_t) voltRMS * currRMS * pf;
  result = (result >> 16);  //Remove scaling factor from the PowerFactor

  //Calculate the integer part
  if (convertToKiloWatts)
  {
    *integerPart = (uint16_t)(result/1000000);  //Converting to kW (and removing
        currentRMS base)
  }
  else
  {
    *integerPart = (uint16_t)(result/1000); //Remove the scaling factor within
        current RMS
  }

  //Calculate the fractional part
  result = (uint64_t) voltRMS * currRMS * pf * 1000;
  result = (result >> 16);

  if (convertToKiloWatts)
  {
    *fractionalPart = (uint16_t)((result/1000000) - ((*integerPart)*1000));
  }
  else
  {
    *fractionalPart = (uint16_t)((result/1000) - ((uint32_t)(*integerPart)*1000)
        );
  }
}

/*! @brief Using the base definition it will normalize and calculate Energy (kWh
    )
 *
 *  @param integerPart - Stores the Integer part of Energy
 *  @param fractionalPart - Stores the fractional part of Energy
 */
static void normaliseEnergy(uint16_t * const integerPart, uint16_t * const
    fractionalPart)
{
  uint64_t result;
  int64_t meterEnergy = Meter_TotalEnergy;  //Copy Global variable into local

  //Calculate Whole part
  result      = meterEnergy * (ENERGY_BASE.base);
  result      = (result >> ENERGY_BASE.n);
  *integerPart = (uint16_t)(result / 10000);  //Scale down again (extra scaling
      factor in base equation)

  //Calculate Fraction
  result      = meterEnergy * 1000 * (ENERGY_BASE.base);
  result      = (result >> ENERGY_BASE.n);
  *fractionalPart  = (uint16_t)((uint64_t)((result / 10000) - ((*integerPart)
```

```
      *1000)));
}

/*! @brief Using the base definition it will normalize and calculate Cost
 *
 *  @param dollars - Stores the $$ of cost
 *  @param cents   - Stores the cc of cost
 */
static void normaliseCost(uint16_t * const dollars, uint8_t * const cents)
{
  uint64_t totalCents;

  totalCents = Meter_TotalEnergyCost * (COST_TOTAL_BASE.base);
  totalCents = (totalCents >> COST_TOTAL_BASE.n);
  totalCents = totalCents / 10000000;             //Account for scaling factor
      in base equation

  //Calculate Dollars
  *dollars  = (uint16_t)(totalCents/100);

  //Calculate Cents
  *cents    = (uint8_t)(totalCents - ((*dollars)*100));
}

/*! @brief Print Time to the terminal.
 *
 */
static void printTime(void)
{
  uint8_t days, hours, minutes, seconds;
  RTC_CalcTime(Meter_Time, &days, &hours, &minutes, &seconds);
  DisplayPrint_Time(days, hours, minutes, seconds);
}

/*! @brief Print Power to the terminal.
 *
 */
static void printPower(void)
{
  uint16_t powerWhole, powerFraction;
  normalisePower(&powerWhole, &powerFraction, bTRUE);
  DisplayPrint_Power(powerWhole, powerFraction);
}

/*! @brief Print energy to the terminal.
 *
 */
static void printEnergy(void)
{
  uint16_t energyWhole, energyFraction;
  normaliseEnergy(&energyWhole, &energyFraction);
  DisplayPrint_Energy(energyWhole, energyFraction);
}

/*! @brief Print cost to the terminal.
 *
 */
```

```
static void printCost(void)
{
  uint16_t dollars;
  uint8_t cents;

  normaliseCost(&dollars, &cents);
  DisplayPrint_Cost(dollars, cents);
}

/*! @brief Call-back for the RTC module.
 *
 *  This will trigger every second, update the user display if it is not dormant
     , and send time information.
 */
static void secondsCallBack(void * nothing)
{
  Meter_IncrementTime();              //Increment the meter timer
  OS_SemaphoreSignal(PrintToTerminal);
}

/*! @brief Responds to a TEST_MODE packet.
 *
 */
static BOOL testModeResponse(void)
{
  //Note we are always running in this mode
  return bTRUE;
}

/*! @brief Responds to a TARIFF packet.
 *
 */
static BOOL tariffResponse(void)
{
  if(Packet_Parameter1 > 0 && Packet_Parameter1 < 4)
  {
    return (Tariff_SetTariff(Packet_Parameter1));
  }

  return bFALSE;
}

/*! @brief Responds to a TIME1 packet.
 *
 */
static BOOL time1Response(void)
{
  uint8_t days, hours, minutes, seconds;
  BOOL responseOk = bFALSE;

  RTC_CalcTime(Meter_Time, &days, &hours, &minutes, &seconds);

  if (Packet_Parameter3 == 1) //User wishes to read seconds and minutes
  {
    responseOk = Packet_Put(TIME1_COMMAND, seconds, minutes, 0);
  }
  else if (Packet_Parameter3 == 2) //User wishes to set seconds and minutes
```

```
  {
    if (Packet_Parameter1 >= 0 && Packet_Parameter1 < 60)
    {
      if (Packet_Parameter2 >= 0 && Packet_Parameter2 < 60)
      {
        Meter_Time = Packet_Parameter1 + (Packet_Parameter2 * 60) + (hours *
            3600) + (days * 86400);
        responseOk = bTRUE;
      }
    }
  }

  return (responseOk);
}

/*! @brief Responds to a TIME2 packet.
 *
 */
static BOOL time2Response(void)
{
  uint8_t days, hours, minutes, seconds;
  BOOL responseOk = bFALSE;

  RTC_CalcTime(Meter_Time, &days, &hours, &minutes, &seconds);

  if (Packet_Parameter3 == 1) //User wishes to read hours and days
  {
    responseOk = Packet_Put(TIME1_COMMAND, seconds, minutes, 0);
  }
  else if (Packet_Parameter3 == 2) //User wishes to set hours and days
  {
    if (Packet_Parameter1 >= 0 && Packet_Parameter1 < 24)
    {
      if (Packet_Parameter2 >= 0 && Packet_Parameter2 < 100)
      {
        Meter_Time = seconds + (minutes * 60) + (Packet_Parameter1 * 3600) + (
            Packet_Parameter2 * 86400);
        responseOk = bTRUE;
      }
    }
  }

  return (responseOk);
}

/*! @brief Responds to a POWER packet.
 *
 */
static BOOL powerResponse(void)
{
  uint16_t powerInteger, powerFraction;
  uint16union_t power;

  normalisePower(&powerInteger, &powerFraction, bFALSE);
  power.l = powerInteger;

  return Packet_Put(POWER_COMMAND, power.s.Lo, power.s.Hi, 0);
```

```
}

/*! @brief Responds to a ENERGY packet.
 *
 */
static BOOL energyResponse(void)
{
  uint16union_t energy;
  uint16_t energyWhole, energyFraction;

  normaliseEnergy(&energyWhole, &energyFraction);
  energy.l = (energyWhole*1000) + energyFraction;  //Apply the extra scaling
      factor of 1000 for displaying energy in a packet

  return (Packet_Put(ENERGY_COMMAND, energy.s.Lo, energy.s.Hi, 0));
}

/*! @brief Responds to a COST packet.
 *
 */
static BOOL costResponse(void)
{
  uint16_t dollars;
  uint8_t cents;
  uint16union_t dollarsUnion;

  normaliseCost(&dollars, &cents);
  dollarsUnion.l = dollars;

  return (Packet_Put(ENERGY_COMMAND, cents, dollarsUnion.s.Lo, dollarsUnion.s.Hi
      ));
}

/*! @brief Responds to a FREQUENCY packet.
 *
 */
static BOOL frequencyResponse(void)
{
  //Waveforms are stuck at 50 Hz
  return (Packet_Put(FREQUENCY_COMMAND, 50, 0, 0));
}

/*! @brief Responds to a VOLT_RMS packet.
 *
 */
static BOOL voltRMSResponse(void)
{
  uint16union_t voltageRMS;
  voltageRMS.l = Meter_VoltageRMS;

  return (Packet_Put(VOLT_RMS_COMMAND, voltageRMS.s.Lo, voltageRMS.s.Hi, 0));
}

/*! @brief Responds to a CURR_RMS packet.
 *
 */
static BOOL currRMSResponse(void)
```

```
{
  uint16union_t currentRMS;
  currentRMS.l = Meter_CurrentRMS;

  return (Packet_Put(CURR_RMS_COMMAND, currentRMS.s.Lo, currentRMS.s.Hi, 0));
}

/*! @brief Responds to a PFACTOR packet.
 *
 */
static BOOL powerFactorResponse(void)
{
  uint16union_t powerFactor;
  uint32_t pf = DAC_PowerFactor();

  powerFactor.l = (uint16_t)((pf*1000) >> 16); //Scale up power factor by 1000

  return (Packet_Put(PFACTOR_COMMAND, powerFactor.s.Lo, powerFactor.s.Hi, 0));
}

/*! @brief Responds to a VOLTAGE_AMP packet.
 *
 */
static BOOL voltageAmplitudeResponse(void)
{
  int16union_t voltageAmplitude;

  voltageAmplitude.s.Lo = Packet_Parameter1;
  voltageAmplitude.s.Hi = Packet_Parameter2;

  if (voltageAmplitude.l >= VOLTAGE_MIN_VALUE && voltageAmplitude.l <=
      VOLTAGE_MAX_VALUE)
  {
    DAC_SetVoltageAmplitude(voltageAmplitude.l);
    return bTRUE;
  }

  return bFALSE;
}

/*! @brief Responds to a CURRENT_AMP packet.
 *
 */
static BOOL currentAmplitudeResponse(void)
{
  int16union_t currentAmplitude;

  currentAmplitude.s.Lo = Packet_Parameter1;
  currentAmplitude.s.Hi = Packet_Parameter2;

  if (currentAmplitude.l >= CURRENT_MIN_VALUE && currentAmplitude.l <=
      CURRENT_MAX_VALUE)
  {
    DAC_SetCurrentAmplitude(currentAmplitude.l);
    return bTRUE;
  }
```

```
  return bFALSE;
}

/*! @brief Responds to a PHASE packet.
 *
 */
static BOOL phaseResponse(void)
{
  if (Packet_Parameter1 >= 0 && Packet_Parameter1 <= 32)
  {
    /* Phase is represented as a series of steps:
     * PhaseStep = 0 -> -90 degrees
     * PhaseStep = 32 -> 90 degrees
     */
    DAC_SetPhaseDifference(Packet_Parameter1);
    return bTRUE;
  }

  return bFALSE;
}

/*! @brief This will decide how to respond to an incoming packet from the PC
 *
 *  @return void
 */
static void packetHandle(void)
{
  //We copy this variable so as to not modify the original when we are removing
      the ACK bit
  uint8_t command = Packet_Command;
  BOOL responseOk = bFALSE;

  //If Packet_Get is successful in receiving a packet from the PC then parse the
       command in packet
  if (Packet_Get())
  {
    //Remove the ACK bit to get the command in 'base' form
    command &= ~(PACKET_ACK_MASK);

    switch (command)
    {
      case TEST_MODE_COMMAND:
        responseOk = testModeResponse();
        break;
      case TARIFF_COMMAND:
        responseOk = tariffResponse();
        break;
      case TIME1_COMMAND:
        responseOk = time1Response();
        break;
      case TIME2_COMMAND:
        responseOk = time2Response();
        break;
      case POWER_COMMAND:
        responseOk = powerResponse();
        break;
      case ENERGY_COMMAND:
```

```
        responseOk = energyResponse();
        break;
      case COST_COMMAND:
        responseOk = costResponse();
        break;
      case FREQUENCY_COMMAND:
        responseOk = frequencyResponse();
        break;
      case VOLT_RMS_COMMAND:
        responseOk = voltRMSResponse();
        break;
      case CURR_RMS_COMMAND:
        responseOk = currRMSResponse();
        break;
      case PFACTOR_COMMAND:
        responseOk = powerFactorResponse();
        break;
      case VOLT_AMP_COMMAND:
        responseOk = voltageAmplitudeResponse();
        break;
      case CURR_AMP_COMMAND:
        responseOk = currentAmplitudeResponse();
        break;
      case PHASE_COMMAND:
        responseOk = phaseResponse();
        break;
    }

    //Check if ACK is required
    if (CHECK_BIT_SET(Packet_Command, 7))
    {
      //In this case, an additional ACK packet is required to be sent if the PC
          requires an acknowledgment
      if (responseOk)
      {
        command |= SET_BIT(7);
      }

      Packet_Put(command, Packet_Parameter1, Packet_Parameter2,
          Packet_Parameter3);
    }
  }
}

/*! @brief Initializes pin information for switch 1.
 *
 *  @return BOOL - TRUE if init was okay
 */
static BOOL switch1Init(void)
{
  Switch1Pressed = OS_SemaphoreCreate(0); //Initialize semaphore relating to the
       switch

  //Clock Gating
  SIM_SCGC5  |= SIM_SCGC5_PORTD_MASK;

  //Set type to GPIO - The pins are set as input by default
```

```
  PORTD_PCR0 &= ~PORT_PCR_MUX_MASK;
  PORTD_PCR0 |= PORT_PCR_MUX(1);

  //Setup interrupts for Pin
  PORTD_PCR0 |= PORT_PCR_ISF_MASK;
  PORTD_PCR0 |= PORT_PCR_IRQC(10);

  //Set Pull-up Resistors
  PORTD_PCR0 |= PORT_PCR_PE_MASK;
  PORTD_PCR0 |= PORT_PCR_PS_MASK;

  //Initialize NVIC
  NVICICPR2  |= (1 << 26); //PORTD
  NVICISER2  |= (1 << 26);

  return (bTRUE);
}

BOOL Interface_Init(void)
{
  BOOL modulesOk = bFALSE;
  BOOL initOk    = bFALSE;

  //Initialize global variables
  DormantDisplayTimerCounter = 1;
  CurrentDisplay = DISPLAY_DORMANT;

  //Initialize relative semaphores
  PrintToTerminal = OS_SemaphoreCreate(0);

  //Initialize DAC (self-test waveforms) with 353.5 V and 7.072A and phase
      difference of 0
  DAC_Init(VOLTAGE_MAX_VALUE, CURRENT_MAX_VALUE, 16);

  //Initialize required modules
  modulesOk = Packet_Init(BAUD_RATE, CPU_BUS_CLK_HZ) && FTM_Init() &&
              RTC_Init(secondsCallBack, 0) && Debounce_Init(cycleDisplayCallBack
                  , 0) && switch1Init();

  if (modulesOk)
  {
    initOk = FTM_Set(&DISPLAY_DORMANT_TIMER);
  }

  return (initOk);
}

void Interface_PrintData(void *pData)
{
  TDisplayState display;
  for (;;)
  {
    OS_SemaphoreWait(PrintToTerminal, 0);

    OS_DisableInterrupts();
    display = CurrentDisplay;
    OS_EnableInterrupts();
```

```
    switch (display)
    {
      case DISPLAY_METER_TIME:
        printTime();
        break;
      case DISPLAY_AVERAGE_POWER:
        printPower();
        break;
      case DISPLAY_TOTAL_ENERGY:
        printEnergy();
        break;
      case DISPLAY_TOTAL_COST:
        printCost();
        break;
    }
  }
}

void Interface_PacketHandle(void *pData)
{
  for (;;)
  {
    packetHandle(); //Handle Packet Data
  }
}

void Interface_CycleDisplay(void *pData)
{
  for (;;)
  {
    OS_SemaphoreWait(Switch1Pressed, 0);

    OS_DisableInterrupts();               //Enter Critical section of Code
    DormantDisplayTimerCounter = 1;       //Reset the TimeoutCounter

    //Advance display state machine
    switch (CurrentDisplay)
    {
      case DISPLAY_DORMANT:
        CurrentDisplay = DISPLAY_METER_TIME;
        break;
      case DISPLAY_METER_TIME:
        CurrentDisplay = DISPLAY_AVERAGE_POWER;
        break;
      case DISPLAY_AVERAGE_POWER:
        CurrentDisplay = DISPLAY_TOTAL_ENERGY;
        break;
      case DISPLAY_TOTAL_ENERGY:
        CurrentDisplay = DISPLAY_TOTAL_COST;
        break;
      case DISPLAY_TOTAL_COST:
        CurrentDisplay = DISPLAY_METER_TIME;
        break;
    }

    FTM_StartTimer(&DISPLAY_DORMANT_TIMER);
```

```
    OS_EnableInterrupts();
  }
}

void __attribute__ ((interrupt)) SW1_ISR(void)
{
  PORTD_ISFR |= PORT_ISFR_ISF(0);        //Write 1 to clear the SW1 interrupt
      flag
  PORTD_PCR0 &= ~PORT_PCR_IRQC_MASK;     //Disable the interrupt on this pin so
      it no longer interrupts

  OS_ISREnter();
  Debounce_Start();
  OS_ISRExit();
}
/* END INTERFACE */
/*!
** @}
*/
```

# 26 interface.h

```
/*! @file
 *
 *  @brief Acts as a human machine interface between the DEM and PC.
 *
 *  This module contains the routines that process user requests.
 *
 *  @author APope
 *  @date 2015-09-30
 */

#ifndef INTERFACE_H
#define INTERFACE_H

#include "types.h"

/*! @brief Initializes the Human machine interface before first use.
 *
 *  @return BOOL - TRUE if initialization was successful.
 */
BOOL Interface_Init(void);

/*! @brief Packet Handle Thread
 *
 *  @param pData - Generic variable to pass data into the Thread.
 *  @return void
 */
void Interface_PacketHandle(void *pData);

/*! @brief Print Data to the terminal Thread
 *
 *  @param pData - Generic variable to pass data into the Thread.
 *  @return void
 */
void Interface_PrintData(void *pData);

/*! @brief Cycle the display mode of the terminal Thread
 *
 *  @param pData - Generic variable to pass data into the Thread.
 *  @return void
 */
void Interface_CycleDisplay(void *pData);

/*! @brief ISR for Switch 1.
 *
 */
void __attribute__ ((interrupt)) SW1_ISR(void);
#endif
```

# 27 math.c

```
/*! @file
 *
 *  @brief Library that performs math operations.
 *
 *  This module contains routines that allows a caller to perform mathematical
     operations.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup MATH_module MATH module documentation
**  @{
*/
/* MODULE MATH */
#include "math.h"

#define NB_ITERATIONS 25

uint32_t Math_SquareRoot(uint32_t num)
{
  uint8_t i;
  uint32_t lastGuess = 0;
  uint32_t estimatedValue = num / 2;

  //Using Newtonian method, find the root
  for (i = 0; i < NB_ITERATIONS; i++)
  {
    if (estimatedValue != 0)
    {
      estimatedValue = ((num / estimatedValue) + estimatedValue) / 2;

      if (estimatedValue == lastGuess)
        break;

      lastGuess = estimatedValue;
    }
    else
    {
      estimatedValue = 0;      //If we encounter a divide by zero error, break
          and return 0
      break;
    }
  }

  return (estimatedValue);
}
/* END MATH */
/*!
** @}
*/
```

# 28 math.h

```
/*! @file
 *
 *  @brief Library that performs math operations.
 *
 *  This module contains routines that allows a caller to perform mathematical
     operations.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef MATH_H
#define MATH_H

#include "types.h"

/*! @brief Calculates the square root of a number.
 *
 *  @param num - Positive unsigned integer to calculate square root value.
 *  @return int32_t - Result
 */
uint32_t Math_SquareRoot(uint32_t num);
#endif
```

# 29 meter.c

```
/*! @file
 *
 *  @brief Maintains usage statistics of the DEM.
 *
 *  This module contains the routines that monitors usage.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup METER_module METER module documentation
**  @{
*/
/* MODULE METER */
#include "sampler.h"
#include "math.h"
#include "meter.h"
#include "tariff.h"
#include "DAC.h"
#include "OS.h"

int64_t Meter_TotalEnergy;
int64_t Meter_TotalEnergyCost;
uint16_t Meter_PowerFactor;
uint16_t Meter_VoltageRMS;
uint16_t Meter_CurrentRMS;
uint32_t Meter_Time;

typedef enum {
  RMS_VOLTAGE,
  RMS_CURRENT
}TRMSType;

static int16_t VoltageSamples[16];      /*!< Contains the samples of the voltage
     waveform for a period. */
static int16_t CurrentSamples[16];      /*!< Contains the samples of the current
     waveform for a period. */
static BOOL PerformingCalculations;     /*!< Indicates that we are currently
    performing calculations on the two sample arrays. */

static OS_ECB *SamplesReady;            /*!< Semaphore to indicate when sample
    data from the 'sampler' module are ready to be read. */
static OS_ECB *RunCalculations;         /*!< Semaphore to indicate when the
    calculation routine should be run. */

/*! @brief Callback function that is invoked when a sample is taken from the
    self-test waveforms.
 *
 */
void sampleCallBack(void * nothing)
{
  OS_SemaphoreSignal(SamplesReady);     //Indicate that samples are ready to be
      read from the 'sampler' module.
}
```

```
/*! @brief Calculates RMS values for voltage and current.
 *
 *  @param sample              - Either a voltage or current sample (these values
      are accumulated within the function).
 *  @param allSamplesAquired   - Once all sample data has been acquired; this
     should be set to TRUE.
 *  @param rmsType             - An identifier for either calculating voltage or
     current RMS.
 */
static void calculateRMS(int16_t sample, BOOL allSamplesAquired, TRMSType
    rmsType)
{
  static uint32_t sumVoltageSamplesSquared = 0;
  static uint32_t sumCurrentSamplesSquared = 0;

  //Square the sample and accumulate
  uint32_t result = (sample * sample);
  switch (rmsType)
  {
    case RMS_VOLTAGE:
      sumVoltageSamplesSquared += (uint32_t)(((uint64_t)result*1000) >> 15); //
          Shift sample base back to original (2^15/1000)
      break;
    case RMS_CURRENT:
      sumCurrentSamplesSquared += (uint32_t)(((uint64_t)result*10) >> 15);   //
          Shift sample base back to original (2^15/10)
      break;
  }

  //By finding the sqrt(sum(samples^2)/16) we are able to find the RMS values
      for the corresponding waveform
  if (allSamplesAquired)
  {
    switch (rmsType)
    {
      case RMS_VOLTAGE:
        //Divide voltage sum by number of samples (always 16); Change 16 to same
            base -> 524 (2^15/1000)
        result = sumVoltageSamplesSquared / 524;
        Meter_VoltageRMS = (uint16_t) Math_SquareRoot(result);
        sumVoltageSamplesSquared = 0;
        break;
      case RMS_CURRENT:
        //Divide current sum by number of samples (always 16); Change 16 to same
            base -> 52428 (2^15/10)
        result = (((uint64_t)sumCurrentSamplesSquared * 1000000) / 52428);  //
            Scale up so result is in RMS mA
        Meter_CurrentRMS = (uint16_t) Math_SquareRoot(result);
        sumCurrentSamplesSquared = 0;
        break;
    }
  }
}

/*! @brief Calculates cost of energy for a particular period.
 *
 *  @param energyForPeriod - The energy calculated for one period.
```

```
 */
static void calculateCost(int32_t energyForPeriod)
{
  Meter_TotalEnergyCost += (int64_t) energyForPeriod * Tariff_GetTariffCost(
      Meter_Time);
}

/*! @brief Calculates energy over a period.
 *
 *  @param voltageSample - Instantaneous voltage sample
 *  @param currentSample - Instantaneous current sample
 *  @param allSamplesCollected - if TRUE, this means that this last sample was
     the final sample over the period (do extra calculations).
 */
static void calculateEnergy(int16_t voltageSample, int16_t currentSample, BOOL
    allSamplesCollected)
{
  static int32_t instantaneousPower = 0;
  instantaneousPower += (voltageSample * currentSample); //Accumulate
      instantaneous power over the period

  if (allSamplesCollected)
  {
    Meter_TotalEnergy += instantaneousPower;  //All samples are collected
        accumulate into total energy (Ts is accounted for in base)
    calculateCost(instantaneousPower);        //Calculate Cost for the period
    instantaneousPower = 0;
  }
}

/*! @brief Runs calculations on meter use statistics.
 *
 *  After all the samples over a period have been collected, this routine is
     invoked.
 */
static void runCalculations(void)
{
  uint8_t i;

  //Loop through the samples to -1 the total sample length
  for(i = 0; i < 15; i++)
  {
    calculateEnergy(VoltageSamples[i], CurrentSamples[i], bFALSE);
    calculateRMS(VoltageSamples[i], bFALSE, RMS_VOLTAGE);
    calculateRMS(CurrentSamples[i], bFALSE, RMS_CURRENT);
  }

  //Calculate the final values, indicating to the functions that all samples
      have been accounted for
  calculateEnergy(VoltageSamples[15], CurrentSamples[15], bTRUE);
  calculateRMS(VoltageSamples[15], bTRUE, RMS_VOLTAGE);
  calculateRMS(CurrentSamples[15], bTRUE, RMS_CURRENT);

  PerformingCalculations = bFALSE;
}

void Meter_IncrementTime(void)
```

```
{
  Meter_Time++;
}

BOOL Meter_Init(void)
{
  Meter_TotalEnergy     = 0;
  Meter_TotalEnergyCost = 0;
  Meter_PowerFactor     = 0;
  Meter_VoltageRMS      = 0;
  Meter_CurrentRMS      = 0;
  Meter_Time            = 0;
  PerformingCalculations   = bFALSE;

  SamplesReady      = OS_SemaphoreCreate(0);
  RunCalculations   = OS_SemaphoreCreate(0);

  return (Sampler_Init(sampleCallBack, 0) && Tariff_Init());
}

void Meter_GetSamples(void *pData)
{
  static uint8_t numberOfSamples = 0;

  for(;;)
  {
    OS_SemaphoreWait(SamplesReady, 0);  //Wait until the sampler module has
        sampled the data

    if (numberOfSamples != 16)
    {
      if (!PerformingCalculations)
      {
        //If we are not currently running calculations on these private global
            arrays, they are available for assignment
        VoltageSamples[numberOfSamples] = Sampler_VoltageSample;
        CurrentSamples[numberOfSamples] = Sampler_CurrentSample;
        numberOfSamples++;
      }
    }
    else
    {
      //We have collected all the required samples, signal other calculation
          thread to run its routines.
      numberOfSamples = 0;
      OS_SemaphoreSignal(RunCalculations);
    }
  }
}

void Meter_RunCalculations(void *pData)
{
  for(;;)
  {
    OS_SemaphoreWait(RunCalculations, 0);   //Wait until the sample arrays have
        been accordingly populated
    PerformingCalculations = bTRUE;
```

```
    runCalculations();
  }
}
/* END METER */
/*!
** @}
*/
```

# 30 meter.h

```
/*! @file
 *
 *  @brief Maintains usage statistics of the DEM.
 *
 *  This module contains the routines that monitors usage.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef METER_H
#define METER_H

#include "types.h"

extern int64_t Meter_TotalEnergy;      /*!< Total recorded energy the DEM has
    measured since power-up. */
extern int64_t Meter_TotalEnergyCost; /*!< Running-cost of the energy the DEM
    has consumed since power-up. */
extern uint16_t Meter_PowerFactor;     /*!< The current power factor. */
extern uint16_t Meter_VoltageRMS;      /*!< Voltage RMS value (V).  Note: This is
     real value (no base) */
extern uint16_t Meter_CurrentRMS;      /*!< Current RMS value (mA). Note: This is
     real value (no base) */
extern uint32_t Meter_Time;            /*!< The amount of time the DEM has been
    running for (seconds). */

/*! @brief Initializes the Meter before first use.
 *
 *  @return BOOL - TRUE if initialization was successful.
 */
BOOL Meter_Init(void);

/*! @brief Thread that performs meter calculations when data becomes available.
 *
 *  @param pData - Generic pointer to pass data into the thread.
 *  @return void
 */
void Meter_RunCalculations(void *pData);

/*! @brief Thread that gets samples from the sampler module at specific
    intervals.
 *
 *  @param pData - Generic pointer to pass data into the thread.
 *  @return void
 */
void Meter_GetSamples(void *pData);

/*! @brief Increments the DEM timer.
 *
 *  @return void
 *  @note Should only be invoked by a seconds interrupt.
 */
void Meter_IncrementTime(void);
#endif
```

# 31 packet.c

```
/*! @file
 *
 *  @brief Routines to implement packet encoding and decoding for the serial
     port.
 *
 *  This contains the functions for implementing the "Tower to PC Protocol" 5-
     byte packets.
 *
 *  @author BAllen, APope
 *  @date 2015-08-15
 */
/*!
**  @addtogroup packet_module packet module documentation
**  @{
*/
/* MODULE packet */
#include "Cpu.h"
#include "UART.h"
#include "packet.h"
#include "OS.h"

TPacket Packet;

static uint8_t Checksum;            /*!< The received packet's Checksum */
static uint8_t PacketState;       /*!< Variable to keep track of the packet
    Finite State Machine */

static uint8_t calculatePacketChecksum(uint8_t command, uint8_t param1, uint8_t
    param2, uint8_t param3);

BOOL Packet_Init(const uint32_t baudRate, const uint32_t moduleClk)
{
  PacketState = 0;
  return (UART_Init(baudRate, moduleClk));
}

BOOL Packet_Get(void)
{
  uint8_t calculatedChecksum;     //The calculated checksum from the command and
       3 parameters of the packet
  uint8_t data;                               //Used as a temporary variable to
      get data from RxFIFO

  if (PacketState != 5)
  {
    if (UART_InChar(&data))
    {
      switch (PacketState)
      {
        case 0:
          Packet_Command = data;
          PacketState++;
          break;
        case 1:
          Packet_Parameter1 = data;
```

```
          PacketState++;
          break;
        case 2:
          Packet_Parameter2 = data;
          PacketState++;
          break;
        case 3:
          Packet_Parameter3 = data;
          PacketState++;
          break;
        case 4:
          Checksum = data;
          PacketState++;
          break;
        default:
          break;
      }
    }
  }
  else if (PacketState == 5)
  {
    //Now we must validate the whole received packet
    calculatedChecksum = calculatePacketChecksum(Packet_Command,
        Packet_Parameter1, Packet_Parameter2, Packet_Parameter3);

    if (calculatedChecksum == Checksum)
    {
      PacketState = 0;
      return (bTRUE);
    }
    else
    {
      //Validation failed, shift the packet bytes to the left, and get a new one
           from RxFIFO the next time this function is called
      Packet_Command = Packet_Parameter1;
      Packet_Parameter1 = Packet_Parameter2;
      Packet_Parameter2 = Packet_Parameter3;
      Packet_Parameter3 = Checksum;

      PacketState--; //Move the Packet state back to state 4
    }
  }

  return (bFALSE);
}

/*! @brief Calculates the checksum of a packet by XOR'ing the preceding 4 bytes.
 *
 *  @param command The packet's command
 *  @param param1 The first parameter of the packet
 *  @param param2 The second parameter of the packet
 *  @param param3 The third parameter of the packet
 *  @return uint8_t - Packets calculated checksum.
 */
static uint8_t calculatePacketChecksum(uint8_t command, uint8_t param1, uint8_t
    param2, uint8_t param3)
{
```

```
  return (((command ^ param1) ^ param2) ^ param3); // XOR operation to determine
      checksum
}

BOOL Packet_Put(const uint8_t command, const uint8_t parameter1, const uint8_t
    parameter2, const uint8_t parameter3)
{
  uint8_t checksum = calculatePacketChecksum(command, parameter1, parameter2,
      parameter3); //Calculate the checksum of the packet
  uint8_t buffer[] = {command, parameter1, parameter2, parameter3, checksum};

  return (UART_PrintDriver(buffer, 5));
}
/* END packet */
/*!
** @}
*/
```

# 32 packet.h

```
/*! @file
 *
 *  @brief Routines to implement packet encoding and decoding for the serial
     port.
 *
 *  This contains the functions for implementing the "Tower to PC Protocol" 5-
     byte packets.
 *
 *  @author PMcL
 *  @date 2015-07-23
 */

#ifndef PACKET_H
#define PACKET_H

// new types
#include "types.h"

// Packet structure
#pragma pack(push)
#pragma pack(1)
typedef struct
{
  uint8_t command;              /*!< The packet's command. */
  union
  {
    struct
    {
      uint8_t parameter1;       /*!< The packet's 1st parameter. */
      uint8_t parameter2;       /*!< The packet's 2nd parameter. */
      uint8_t parameter3;       /*!< The packet's 3rd parameter. */
    } separate;
    struct
    {
      uint16_t parameter12;
      uint8_t parameter3;
    } combined12;
    struct
    {
      uint8_t paramater1;
      uint16_t parameter23;
    } combined23;
  } parameters;
} TPacket;
#pragma pack(pop)

#define Packet_Command     Packet.command
#define Packet_Parameter1  Packet.parameters.separate.parameter1
#define Packet_Parameter2  Packet.parameters.separate.parameter2
#define Packet_Parameter3  Packet.parameters.separate.parameter3
#define Packet_Parameter12 Packet.parameters.combined12.parameter12
#define Packet_Parameter23 Packet.parameters.combined23.parameter23

extern TPacket Packet;
```

```
// Acknowledgment bit mask
extern const uint8_t PACKET_ACK_MASK;

/*! @brief Initializes the packets by calling the initialization routines of the
     supporting software modules.
 *
 *  @param baudRate The desired baud rate in bits/sec.
 *  @param moduleClk The module clock rate in Hz.
 *  @return BOOL - TRUE if the packet module was successfully initialized.
 */
BOOL Packet_Init(const uint32_t baudRate, const uint32_t moduleClk);

/*! @brief Attempts to get a packet from the received data.
 *
 *  @return BOOL - TRUE if a valid packet was received.
 */
BOOL Packet_Get(void);

/*! @brief Builds a packet and places it in the transmit FIFO buffer.
 *
 *  @return BOOL - TRUE if a valid packet was sent.
 */
BOOL Packet_Put(const uint8_t command, const uint8_t parameter1, const uint8_t
    parameter2, const uint8_t parameter3);

#endif
```

# 33 sampler.c

```
/*! @file
 *
 *  @brief Retrieves samples from the DAC (Digital to Analog Converter).
 *
 *  This module contains the routines that take samples from the current and
     voltage waveforms.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup SAMPLER_module SAMPLER module documentation
**  @{
*/
/* MODULE SAMPLER */
#include "Cpu.h"
#include "DAC.h"
#include "PIT.h"
#include "sampler.h"

int16_t Sampler_VoltageSample;
int16_t Sampler_CurrentSample;

static void (*UserFunction)(void *);
static void *UserArguments;

void samplerCallBack(void * nothing)
{
  //Triggers on the PIT every 800Hz
  static uint16_t voltageSamplePos = 0;  //Will start at sine(0) of the LUT
  static uint16_t currentSamplePos = 0;

  DAC_GetSample(&Sampler_VoltageSample, &Sampler_CurrentSample, voltageSamplePos
      , currentSamplePos);

  //We traverse the array at multiples of 4 (to get the desired 16 samples
      within the 64).
  voltageSamplePos = (voltageSamplePos + 4) % 64;
  currentSamplePos = (currentSamplePos + 4) % 64;

  //Invoke the User callback function
  UserFunction(UserArguments);
}

BOOL Sampler_Init(void (*userFunction)(void *), void *userArguments)
{
  //16 samples per period; -> 50Hz waveform -> sampling period of: (800Hz or
      1.25mS)
  BOOL initOk = bFALSE;

  if (PIT_Init(CPU_BUS_CLK_HZ, samplerCallBack, 0))
  {
    UserFunction = userFunction;
    UserArguments = userArguments;
```

```
    PIT_Set(1250000, bTRUE);
    return (bTRUE);
  }

  return (bFALSE);
}
/* END SAMPLER */
/*!
** @}
*/
```

# 34 sampler.h

```
/*! @file
 *
 *  @brief Retrieves samples from the DAC (Digital to Analog Converter).
 *
 *  This module contains the routines that take samples from the current and
     voltage waveforms.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef SAMPLER_H
#define SAMPLER_H

#include "types.h"

extern int16_t Sampler_VoltageSample; /*!< The most recent voltage sample. */
extern int16_t Sampler_CurrentSample; /*!< The most recent current sample. */

/*! @brief Initializes the Sampler before first use.
 *
 *  @param userFunction  - Callback function that invokes at the same frequency
     as the sampler.
 *  @param userArguments - The arguments to the call-back function.
 *  @return BOOL - TRUE if initialization was successful.
 */
BOOL Sampler_Init(void (*userFunction)(void *), void *userArguments);
#endif
```

# 35 tariff.c

```
/*! @file
 *
 *  @brief Controls the tariffs of the DEM.
 *
 *  Provides tariff information depending on user selection.
 *
 *  @author APope
 *  @date 2015-09-30
 */
/*!
**  @addtogroup TARIFF_module TARIFF module documentation
**  @{
*/
/* MODULE TARIFF */
#include "tariff.h"
#include "Flash.h"
#include "RTC.h"

volatile uint8_t *NvTariffMode;     /*!< Address of the Tariff Mode in Flash
    Memory. */

static const int32_t TARRIF_1_PEAK        = 22235; //(1000)*22.235; -> cents/kWh
static const int32_t TARRIF_1_SHOULDER    = 4400;  //(1000)*4.400;  -> cents/kWh
static const int32_t TARRIF_1_OFF_PEAK    = 2109;  //(1000)*2.109;  -> cents/kWh

static const int32_t TARRIF_2             = 1713;  //(1000)*1.713;  -> cents/kWh
static const int32_t TARRIF_3             = 4100;  //(1000)*4.100;  -> cents/kWh

BOOL Tariff_Init(void)
{
  BOOL initOk = bFALSE;

  //Attempt to initialize flash and check that the tariff is not set to default
  if (Flash_Init())
  {
    if (Flash_AllocateVar((volatile void **)&NvTariffMode, sizeof((*NvTariffMode
        ))))
    {
      if ((*NvTariffMode) == 0xFF)
      {
        //Default mode will be tariff mode 1
        initOk = Flash_Write8(NvTariffMode, TARIFF_MODE_1);
      }
      else
      {
        initOk = bTRUE;
      }
    }
  }

  return (initOk);
}

BOOL Tariff_SetTariff(TTariff tariffMode)
{
```

```
  return (Flash_Write8(NvTariffMode, tariffMode));
}

int32_t getTariffBasedOnTime(uint32_t time)
{
  int32_t tariffCost = 0;
  uint8_t days, hours, minutes, seconds;

  RTC_CalcTime(time, &days, &hours, &minutes, &seconds);

  if (hours > 14 && hours < 20)
  {
    //We are in Peak
    tariffCost = TARRIF_1_PEAK;
  }
  else if ((hours > 7 && hours < 14) || (hours > 20 && hours < 22))
  {
    //We are in shoulder
    tariffCost = TARRIF_1_SHOULDER;
  }
  else
  {
    //We are in Off-Peak
    tariffCost = TARRIF_1_OFF_PEAK;
  }

  return (tariffCost);
}

int32_t Tariff_GetTariffCost(uint32_t time)
{
  int32_t tariffCost = 0;

  switch ((*NvTariffMode))
  {
    case TARIFF_MODE_1:
      tariffCost = getTariffBasedOnTime(time);
      break;
    case TARIFF_MODE_2:
      tariffCost = TARRIF_2;
      break;
    case TARIFF_MODE_3:
      tariffCost = TARRIF_3;
      break;
  }

  return (tariffCost);
}
/* END TARIFF */
/*!
** @}
*/
```

# 36 tariff.h

```
/*! @file
 *
 *  @brief Controls the tariffs of the DEM.
 *
 *  Provides tariff information depending on user selection.
 *
 *  @author APope
 *  @date 2015-09-30
 */
#ifndef TARIFF_H
#define TARIFF_H

#include "types.h"

typedef enum
{
  TARIFF_MODE_1 = 1,
  TARIFF_MODE_2 = 2,
  TARIFF_MODE_3 = 3
}TTariff;

/*! @brief Initializes the tariff module before first use.
 *
 *  @return BOOL - TRUE if initialization was successful.
 */
BOOL Tariff_Init(void);

/*! @brief Sets the tariff mode.
 *
 *  @param tariffMode - The tariff mode to be set.
 *  @return BOOL - TRUE if write to flash was successful.
 */
BOOL Tariff_SetTariff(TTariff tariffMode);

/*! @brief Gets the current price of electricity (kWh).
 *
 *  @param time - Used when in tariff mode 2, calculating tariff based on time.
 *  @return int32_t - Tariff (cents per kWh).
 */
int32_t Tariff_GetTariffCost(uint32_t time);
#endif
```

# 37 types.h

```
/*! @file
 *
 *  @brief Declares new types.
 *
 *  This contains types that are especially useful for the Tower to PC Protocol.
 *
 *  @author PMcL
 *  @date 2015-07-23
 */

#ifndef TYPES_H
#define TYPES_H

#include <stdint.h>

// Unions to efficiently access hi and lo parts of integers and words
typedef union
{
  int16_t l;
  struct
  {
    int8_t Lo;
    int8_t Hi;
  } s;
} int16union_t;

typedef union
{
  uint16_t l;
  struct
  {
    uint8_t Lo;
    uint8_t Hi;
  } s;
} uint16union_t;

// Union to efficiently access hi and lo parts of a long integer
typedef union
{
  uint32_t l;
  struct
  {
    uint16_t Lo;
    uint16_t Hi;
  } s;
} uint32union_t;

// Union to efficiently access hi and lo parts of a "phrase" (8 bytes)
typedef union
{
  uint64_t l;
  struct
  {
    uint32_t Lo;
    uint32_t Hi;
```

```
  } s;
} uint64union_t;

// Union to efficiently access individual bytes of a float
typedef union
{
  float d;
  struct
  {
    uint16union_t dLo;
    uint16union_t dHi;
  } dParts;
} TFloat;

/*! Boolean definition that includes type and value */
typedef enum
{
  bFALSE = 0,    /*!< Boolean false - always 0*/
  bTRUE = 1      /*!< Boolean true - always 1 */
} BOOL;

/*! State definition of a button/switch/capacitive touch plate */
typedef enum
{
  LOGIC_0,
  LOGIC_1
} TButtonState;

#endif
```