Appendix A – PIC32 Code

Contents

1. Header Files

configbits.h

```
/*
 * @file configbits.h
 * @author Mike Schafer
 * @brief Configuration settings for PIC32 microcontroller
 *        2016 board with 8 MHz resonator
 */

#ifndef CONFIGBITS_H
#define    CONFIGBITS_H

/*
 using external osc
 peripheral clock = at 40 MHz (80 MHz/2)
 */

#pragma config FNOSC = FRCPLL     // Oscillator selection
#pragma config POSCMOD = HS       // Primary oscillator mode
#pragma config FPLLIDIV = DIV_2   // PLL input divider (8 -> 4)
#pragma config FPLLMUL = MUL_20   // PLL multiplier (4x20 = 80)
#pragma config FPLLODIV = DIV_1   // PLL output divider
#pragma config FPBDIV = DIV_1     // Peripheral bus clock divider (80/1 = 80 mhz)
#pragma config FSOSCEN = OFF      // Secondary oscillator enable

/* Clock control settings */
#pragma config IESO = ON          // Internal/external clock switchover
#pragma config FCKSM = CSECME     // Clock switching (CSx)/Clock monitor (CMx)
#pragma config OSCIOFNC = OFF     // Clock output on OSCO pin enable

/* USB Settings */
//#pragma config UPLLEN = OFF     // USB PLL enable
//#pragma config UPLLIDIV = DIV_2 // USB PLL input divider
//#pragma config FVBUSONIO = OFF  // VBUS pin control
//#pragma config FUSBIDIO = OFF   // USBID pin control

/* Other Peripheral Device settings */
#pragma config FWDTEN = OFF         // Watchdog timer enable
#pragma config WDTPS = PS4096       // Watchdog timer post-scaler
#pragma config FSRSSEL = PRIORITY_7 // SRS interrupt priority
#pragma config DEBUG = ON

#pragma config ICESEL = ICS_PGx1    // ICE pin selection


#endif     /* CONFIGBITS_H */
```

---

deca_decive_api.h

```
/*! -----------------------------------------------------------
 * @file deca_device_api.h
 * @brief DW1000 API Functions
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 3.3.17
 */

#ifndef _DECA_DEVICE_API_H_
#define _DECA_DEVICE_API_H_

#ifdef __cplusplus
extern "C" {
#endif


#ifndef uint8
#ifndef _DECA_UINT8_
#define _DECA_UINT8_
typedef unsigned char uint8;
#endif
#endif

#ifndef uint16
#ifndef _DECA_UINT16_
#define _DECA_UINT16_
typedef unsigned short uint16;
#endif
#endif

#ifndef uint32
#ifndef _DECA_UINT32_
#define _DECA_UINT32_
typedef unsigned long uint32;
#endif
#endif

#ifndef int8
#ifndef _DECA_INT8_
#define _DECA_INT8_
typedef signed char int8;
#endif
#endif

#ifndef int16
#ifndef _DECA_INT16_
#define _DECA_INT16_
typedef signed short int16;
#endif
#endif

#ifndef int32
#ifndef _DECA_INT32_
#define _DECA_INT32_
typedef signed long int32;
```

```
#endif
#endif

#define DWT_SUCCESS     (0)
#define DWT_ERROR       (-1)

#define DWT_TIME_UNITS  (1.0/499.2e6/128.0) //!< = 15.65e-12 s
#define DWT_DEVICE_ID   (0xDECA0130)        //!< DW1000 MP device ID

/* Constants for selecting the bit rate for data TX (and RX)
 * These are defined for write (with just a shift) the TX_FCTRL register */
#define DWT_BR_110K     0   //!< UWB bit rate 110 kbits/s
#define DWT_BR_850K     1   //!< UWB bit rate 850 kbits/s
#define DWT_BR_6M8      2   //!< UWB bit rate 6.8 Mbits/s

/* Constants for specifying the (Nominal) mean Pulse Repetition Frequency
 * These are defined for direct write (with a shift if necessary) to CHAN_CTRL
and TX_FCTRL regs */
#define DWT_PRF_16M     1   //!< UWB PRF 16 MHz
#define DWT_PRF_64M     2   //!< UWB PRF 64 MHz

/* Constants for specifying Preamble Acquisition Chunk (PAC) Size in symbols */
#define DWT_PAC8        0   //!< PAC  8 (recommended for RX of preamble length
                             128 and below
#define DWT_PAC16       1   //!< PAC 16 (recommended for RX of preamble length
                             256
#define DWT_PAC32       2   //!< PAC 32 (recommended for RX of preamble length
                             512
#define DWT_PAC64       3   //!< PAC 64 (recommended for RX of preamble length
                             1024 and up

/* Constants for specifying TX Preamble length in symbols
 * These are defined to allow them be directly written into byte 2 of the
 * TX_FCTRL register
 * (i.e. a four bit value destined for bits 20..18 but shifted left by 2 for byte
 * alignment) */
#define DWT_PLEN_4096   0x0C    // Standard preamble length 4096 symbols
#define DWT_PLEN_2048   0x28    // Non-standard preamble length 2048 symbols
#define DWT_PLEN_1536   0x18    // Non-standard preamble length 1536 symbols
#define DWT_PLEN_1024   0x08    // Standard preamble length 1024 symbols
#define DWT_PLEN_512    0x34    // Non-standard preamble length 512 symbols
#define DWT_PLEN_256    0x24    // Non-standard preamble length 256 symbols
#define DWT_PLEN_128    0x14    // Non-standard preamble length 128 symbols
#define DWT_PLEN_64     0x04    // Standard preamble length 64 symbols

#define DWT_SFDTOC_DEF      0x1041  // default SFD timeout value

#define DWT_PHRMODE_STD     0x0     // standard PHR mode
#define DWT_PHRMODE_EXT     0x3     // DW proprietary extended frames PHR mode

/* Defined constants for "mode" bitmask parameter passed into dwt_starttx()
 * function. */
#define DWT_START_TX_IMMEDIATE      0
```

```
#define DWT_START_TX_DELAYED        1
#define DWT_RESPONSE_EXPECTED       2

#define DWT_START_RX_IMMEDIATE  0
#define DWT_START_RX_DELAYED    1   // Set up delayed RX, if "late" error
                                   triggers, then the RX will be enabled
                                   immediately
#define DWT_IDLE_ON_DLY_ERR     2   // If delayed RX failed due to "late" error
                                   then if this flag is set the RX will
                                    //not be re-enabled immediately, and device
                                   will be in IDLE when function exits
#define DWT_NO_SYNC_PTRS        4   // Do not try to sync IC side and Host side
                                   buffer pointers when enabling RX. This is used
                                    // to perform manual RX re-enabling when
                                   receiving a frame in double buffer mode.

/* Defined constants for "mode" bit field parameter passed to dwt_setleds()
 * function. */
#define DWT_LEDS_DISABLE     0x00
#define DWT_LEDS_ENABLE      0x01
#define DWT_LEDS_INIT_BLINK  0x02

/* Frame filtering configuration options */
#define DWT_FF_NOTYPE_EN            0x000           // no frame types allowed (FF
disabled)
#define DWT_FF_COORD_EN             0x002           // behave as coordinator (can
                                                   receive frames with no dest
                                                   address (PAN ID has to match))
#define DWT_FF_BEACON_EN            0x004           // beacon frames allowed
#define DWT_FF_DATA_EN              0x008           // data frames allowed
#define DWT_FF_ACK_EN               0x010           // ack frames allowed
#define DWT_FF_MAC_EN               0x020           // mac control frames allowed
#define DWT_FF_RSVD_EN              0x040           // reserved frame types
                                                   allowed

/* DW1000 interrupt events */
#define DWT_INT_TFRS            0x00000080          // frame sent
#define DWT_INT_LDED            0x00000400          // micro-code has finished
                                                    execution
#define DWT_INT_RFCG            0x00004000          // frame received with good
                                                    CRC
#define DWT_INT_RPHE            0x00001000          // receiver PHY header error
#define DWT_INT_RFCE            0x00008000          // receiver CRC error
#define DWT_INT_RFSL            0x00010000          // receiver sync loss error
#define DWT_INT_RFTO            0x00020000          // frame wait timeout
#define DWT_INT_RXOVRR          0x00100000          // receiver overrun
#define DWT_INT_RXPTO           0x00200000          // preamble detect timeout
#define DWT_INT_SFDT            0x04000000          // SFD timeout
#define DWT_INT_ARFE            0x20000000          // frame rejected (due to
                                                   frame filtering configuration)

/* DW1000 INIT configuration parameters */
#define DWT_LOADUCODE     0x1
```

```
#define DWT_LOADNONE      0x0

/* TX/RX call-back data */
typedef struct{
} dwt_cb_data_t;

/* Call-back type for all events */
typedef void (*dwt_cb_t)(const dwt_cb_data_t *);

/*! ------------------------------------------------------------------------------
 * Structure typedef: dwt_config_t
 * Structure for setting device configuration via dwt_configure() function
 */
typedef struct {
    uint8 chan;           //!< channel number {1, 2, 3, 4, 5, 7 }
    uint8 prf;            //!< Pulse Repetition Frequency {DWT_PRF_16M or
                           DWT_PRF_64M}
    uint8 txPreambLength; //!< DWT_PLEN_64..DWT_PLEN_4096
    uint8 rxPAC;          //!< Acquisition Chunk Size (Relates to RX preamble
                           length)
    uint8 txCode;         //!< TX preamble code
    uint8 rxCode;         //!< RX preamble code
    uint8 nsSFD;          //!< Boolean should we use non-standard SFD for better
                           performance
    uint8 dataRate;       //!< Data Rate {DWT_BR_110K, DWT_BR_850K or DWT_BR_6M8}
    uint8 phrMode;        //!< PHR mode {0x0 - standard DWT_PHRMODE_STD, 0x3 -
                           extended frames DWT_PHRMODE_EXT}
    uint16 sfdTO;         //!< SFD timeout value (in symbols)
} dwt_config_t;

typedef struct {
    uint16      maxNoise;          // LDE max value of noise
    uint16      firstPathAmp1;     // Amplitude at floor(index FP) + 1
    uint16      stdNoise;          // Standard deviation of noise
    uint16      firstPathAmp2;     // Amplitude at floor(index FP) + 2
    uint16      firstPathAmp3;     // Amplitude at floor(index FP) + 3
    uint16      maxGrowthCIR;      // Channel Impulse Response max growth CIR
    uint16      rxPreamCount;      // Count of preamble symbols accumulated
    uint16      firstPath;         // First path index (10.6 bits fixed point
                                    integer)
}dwt_rxdiag_t;

typedef struct {
    // all of the below are mapped to a 12-bit register in DW1000
    uint16 PHE;                    //number of received header errors
    uint16 RSL;                    //number of received frame sync loss events
    uint16 CRCG;                   //number of good CRC received frames
    uint16 CRCB;                   //number of bad CRC (CRC error) received
                                    frames
    uint16 ARFE;                   //number of address filter errors
    uint16 OVER;                   //number of receiver overflows (used in double
                                    buffer mode)
    uint16 SFDTO;                  //SFD timeouts
    uint16 PTO;                     //Preamble timeouts
    uint16 RTO;                     //RX frame wait timeouts
    uint16 TXF;                     //number of transmitted frames
    uint16 HPW;                     //half period warn
    uint16 TXW;                     //power up warn
} dwt_deviceentcnts_t;


/*! ------------------------------------------------------------------------------------
 * @fn dwt_readdevid()
 * @originates deca_device.c
 */
uint32 dwt_readdevid(void);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_initialise()
 * @originates deca_device.c
 */
int dwt_initialise(uint16 config);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_configure()
 * @originates deca_device.c
 */
void dwt_configure(dwt_config_t *config);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setrxantennadelay()
 * @originates deca_device.c
 */
void dwt_setrxantennadelay(uint16 antennaDly);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_settxantennadelay()
 * @originates deca_device.c
 */
void dwt_settxantennadelay(uint16 antennaDly);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_writetxdata()
 * @originates deca_device.c
 */
int dwt_writetxdata(uint16 txFrameLength, uint8 *txFrameBytes, uint16
    txBufferOffset);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_writetxfctrl()
 * @originates deca_device.c
 */
void dwt_writetxfctrl(uint16 txFrameLength, uint16 txBufferOffset, int ranging);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_starttx()
 * @originates deca_device.c
 */
int dwt_starttx(uint8 mode);

/*! ------------------------------------------------------------------------------
 * @fn dwt_setdelayedtrxtime()
 * @originates deca_device.c
 */
void dwt_setdelayedtrxtime(uint32 starttime);

/*! ------------------------------------------------------------------------------
 * @fn dwt_readtxtimestamp()
 * @originates deca_device.c
 */
void dwt_readtxtimestamp(uint8 * timestamp);

/*! ------------------------------------------------------------------------------
 * @fn dwt_readtxtimestamplo32()
 * @originates deca_device.c
 */
uint32 dwt_readtxtimestamplo32(void);

/*! ------------------------------------------------------------------------------
 * @fn dwt_readrxtimestamp()
 * @originates deca_device.c
 */
void dwt_readrxtimestamp(uint8 * timestamp);

/*! ------------------------------------------------------------------------------
 * @fn dwt_readrxtimestamplo32()
 * @originates deca_device.c
 */
uint32 dwt_readrxtimestamplo32(void);

/*! ------------------------------------------------------------------------------
 * @fn dwt_readsystime()
 * @originates deca_device.c
 */
void dwt_readsystime(uint8 * timestamp);

/*! ------------------------------------------------------------------------------
 * @fn dwt_syncrxbufptrs()
 * @originates deca_device.c
 */
void dwt_syncrxbufptrs(void);

/*! ------------------------------------------------------------------------------
 * @fn dwt_rxenable()
 * @originates deca_device.c
 */
int dwt_rxenable(int mode);

/*! ------------------------------------------------------------------------------
```

```
 * @fn dwt_setrxtimeout()
 * @originates deca_device.c
 */
void dwt_setrxtimeout(uint16 time);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setpanid()
 * @originates deca_device.c
 */
void dwt_setpanid(uint16 panID);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setaddress16()
 * @originates deca_device.c
 */
void dwt_setaddress16(uint16 shortAddress);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_seteui()
 * @originates deca_device.c
 */
void dwt_seteui(uint8 *eui64);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_geteui()
 * @originates deca_device.c
 */
void dwt_geteui(uint8 *eui64);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_enableframefilter()
 * @originates deca_device.c
 */
void dwt_enableframefilter(uint16 bitmask);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setrxaftertxdelay()
 * @originates deca_device.c
 */
void dwt_setrxaftertxdelay(uint32 rxDelayTime);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_rxreset()
 * @originates deca_device.c
 */
void dwt_rxreset(void);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_softreset()
 * @originates deca_device.c
 */
void dwt_softreset(void);
```

```
/*! ------------------------------------------------------------------------
 * @fn dwt_readrxdata()
 * @originates deca_device.c
 */
void dwt_readrxdata(uint8 *buffer, uint16 length, uint16 rxBufferOffset);

/*! ------------------------------------------------------------------------
 * @fn dwt_setxtaltrim()
 * @originates deca_device.c
 */
void dwt_setxtaltrim(uint8 value);

/*! ------------------------------------------------------------------------
 * @fn dwt_writetodevice()
 * @originates deca_device.c
 */
void dwt_writetodevice
(
    uint16 recordNumber,   // input parameter - ID of register file or buffer
                           being accessed
    uint16 index,          // input parameter - byte index into register file or
                           buffer being accessed
    uint32 length,         // input parameter - number of bytes being written
    const uint8 *buffer    // input parameter - pointer to buffer containing the
                           'length' bytes to be written
);

/*! ------------------------------------------------------------------------
 * @fn dwt_readfromdevice()
 * @originates deca_device.c
 */
void dwt_readfromdevice
(
    uint16 recordNumber,       // input parameter - ID of register file or buffer
                               being accessed
    uint16 index,              // input parameter - byte index into register file
                               or buffer being accessed
    uint32 length,             // input parameter - number of bytes being read
    uint8  *buffer             // input parameter - pointer to buffer in which to
                               return the read data.
);

/*! ------------------------------------------------------------------------
 * @fn dwt_read32bitoffsetreg()
 * @originates deca_device.c
 */
uint32 dwt_read32bitoffsetreg(int regFileID, int regOffset);

/*! ------------------------------------------------------------------------
 * @fn dwt_write32bitoffsetreg()
 * @originates deca_device.c
 */
void dwt_write32bitoffsetreg(int regFileID, int regOffset, uint32 regval);
```

```c
#define dwt_write32bitreg(x,y)  dwt_write32bitoffsetreg(x,0,y)
#define dwt_read32bitreg(x)     dwt_read32bitoffsetreg(x,0)

/*! ------------------------------------------------------------------------------------
 * @fn dwt_read16bitoffsetreg()
 * @originates deca_device.c
 */
uint16 dwt_read16bitoffsetreg(int regFileID, int regOffset);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_write16bitoffsetreg()
 * @originates deca_device.c
 */
void dwt_write16bitoffsetreg(int regFileID, int regOffset, uint16 regval);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_read8bitoffsetreg()
 * @originates deca_device.c
 */
uint8 dwt_read8bitoffsetreg(int regFileID, int regOffset);

/*! ------------------------------------------------------------------------------------
 * @fn dwt_write8bitoffsetreg()
 * @originates deca_device.c
 */
void dwt_write8bitoffsetreg(int regFileID, int regOffset, uint8 regval);


/*! ------------------------------------------------------------------------------------
 * @fn writetospi()
 * @originates deca_spi.c
 */
int writetospi(uint16 headerLength, const uint8 *headerBuffer, uint32 bodylength,
   const uint8 *bodyBuffer);

/*! ------------------------------------------------------------------------------------
 * @fn readfromspi()
 * @originates deca_spi.c
 */
int readfromspi(uint16 headerLength, const uint8 *headerBuffer, uint32
   readlength, uint8 *readBuffer);

/*! ------------------------------------------------------------------------------------
 * @fn deca_sleep()
 * @originates deca_sleep.c
 */
void deca_sleep(unsigned int time_ms);

#ifdef __cplusplus
}
#endif
```

```
#endif /* _DECA_DEVICE_API_H_ */
```

---

deca_param_types.h

```
/*! ------------------------------------------------------------------------------
 * @file deca_param_types.h
 * @brief Decawave general type definitions for configuration structures
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */
#ifndef _DECA_PARAM_TYPES_H_
#define _DECA_PARAM_TYPES_H_

#ifdef __cplusplus
extern "C" {
#endif
#include "deca_types.h"

#define NUM_BR 3
#define NUM_PRF 2
#define NUM_PACS 4
#define NUM_BW 2            //2 bandwidths are supported
#define NUM_SFD 2           //supported number of SFDs - standard = 0, non-
                             standard = 1
#define NUM_CH 6            //supported channels are 1, 2, 3, 4, 5, 7
#define NUM_CH_SUPPORTED 8  //supported channels are '0', 1, 2, 3, 4, 5, '6', 7
#define PCODES 25           //supported preamble codes

typedef struct {
    uint32 lo32;
    uint16 target[NUM_PRF];
} agc_cfg_struct;

extern const agc_cfg_struct agc_config;
/* SFD threshold settings for 110k, 850k, 6.8Mb standard and non-standard */
extern const uint16 sftsh[NUM_BR][NUM_SFD];
extern const uint16 dtune1[NUM_PRF];

#define XMLPARAMS_VERSION   (1.17f)

extern const uint32 fs_pll_cfg[NUM_CH];
extern const uint8 fs_pll_tune[NUM_CH];
extern const uint8 rx_config[NUM_BW];
extern const uint32 tx_config[NUM_CH];
extern const uint8 dwnsSFDlen[NUM_BR]; //length of SFD for each of the bitrates
extern const uint32 digital_bb_config[NUM_PRF][NUM_PACS];
extern const uint8 chan_idx[NUM_CH_SUPPORTED];

#define PEAK_MULTPLIER  (0x60) //3 -> (0x3 * 32) & 0x00E0
#define N_STD_FACTOR    (13)
#define LDE_PARAM1      (PEAK_MULTPLIER | N_STD_FACTOR)
#define LDE_PARAM3_16   (0x1607)
#define LDE_PARAM3_64   (0x0607)
extern const uint16 lde_replicaCoeff[PCODES];

#ifdef __cplusplus
}
#endif

#endif
```

---

deca_regs.h

```
/*! ------------------------------------------------------------------------------
 * @file deca_regs.h
 * @brief DW1000 Register Definitions
 *        This file supports assembler and C development for DW1000 enabled devices
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 3.3.17
 */

#ifndef _DECA_REGS_H_
#define _DECA_REGS_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "deca_version.h"

/****************************************************************************//
** @brief Bit definitions for register DEV_ID
**/
#define DEV_ID_ID               0x00            // Device ID register, includes
revision info (0xDECA0130)

/****************************************************************************//
** @brief Bit definitions for register EUI_64
**/
#define EUI_64_ID               0x01            // IEEE Extended Unique
Identifier (63:0)
#define EUI_64_OFFSET           0x00
#define EUI_64_LEN              (8)

/***************************************************************************//**
 * @brief Bit definitions for register PANADR
**/
#define PANADR_ID               0x03            // PAN ID (31:16) and Short
Address (15:0)
/* mask and shift */
#define PANADR_SHORT_ADDR_OFFSET 0              // In bytes
#define PANADR_SHORT_ADDR_MASK  0x0000FFFFUL    // Short Address
#define PANADR_PAN_ID_OFFSET     2              // In bytes
```

```
#define PANADR_PAN_ID_MASK       0xFFFF00F0UL    // PAN Identifier

/****************************************************************************//
** @brief Bit definitions for register SYS_CFG
**/
#define SYS_CFG_ID               0x04            // System Configuration (31:0)
/* mask and shift */
#define SYS_CFG_MASK             0xF047FFFFUL    // access mask to SYS_CFG_ID
#define SYS_CFG_FF_ALL_EN        0x000001FEUL    // Frame filtering options all
                                                 frames allowed
/* offset 0 */
#define SYS_CFG_FFE              0x00000001UL    // Frame Filtering Enable. This
                                                 bit enables the frame filtering
                                                 functionality
#define SYS_CFG_FFBC             0x00000002UL    // Frame Filtering Behave as a
                                                 Co-ordinator
#define SYS_CFG_FFAB             0x00000004UL    // Frame Filtering Allow Beacon
                                                 frame reception
#define SYS_CFG_FFAD             0x00000008UL    // Frame Filtering Allow Data
                                                 frame reception
#define SYS_CFG_FFAA             0x00000010UL    // Frame Filtering Allow
                                                 Acknowledgment frame reception
#define SYS_CFG_FFAM             0x00000020UL    // Frame Filtering Allow MAC
                                                 command frame reception
#define SYS_CFG_FFAR             0x00000040UL    // Frame Filtering Allow Reserved
                                                 frame types
#define SYS_CFG_FFA4             0x00000080UL    // Frame Filtering Allow frames
                                                 with frame type field of 4,
                                                 (binary 100)
/* offset 8 */
#define SYS_CFG_FFA5             0x00000100UL    // Frame Filtering Allow frames
                                                 with frame type field of 5,
                                                 (binary 101)
#define SYS_CFG_HIRQ_POL         0x00000200UL    // Host interrupt polarity
#define SYS_CFG_SPI_EDGE         0x00000400UL    // SPI data launch edge
#define SYS_CFG_DIS_FCE          0x00000800UL    // Disable frame check error
                                                 handling
#define SYS_CFG_DIS_DRXB         0x00001000UL    // Disable Double RX Buffer
#define SYS_CFG_DIS_PHE          0x00002000UL    // Disable receiver abort on PHR
                                                 error
#define SYS_CFG_DIS_RSDE         0x00004000UL    // Disable Receiver Abort on RSD
                                                 error
#define SYS_CFG_FCS_INIT2F       0x00008000UL    // initial seed value for the FCS
                                                 generation and checking function
/* offset 16 */
#define SYS_CFG_PHR_MODE_SHFT    16
#define SYS_CFG_PHR_MODE_00      0x00000000UL    // Standard Frame mode
#define SYS_CFG_PHR_MODE_11      0x00030000UL    // Long Frames mode
#define SYS_CFG_DIS_STXP         0x00040000UL    // Disable Smart TX Power control
#define SYS_CFG_RXM110K          0x00400000UL    // Receiver Mode 110 kbps data rate
/* offset 24 */
#define SYS_CFG_RXWTOE           0x10000000UL    // Receive Wait Timeout Enable.
```

```
#define SYS_CFG_RXAUTR          0x20000000UL    // Receiver Auto-Re-enable. This
                                                bit is used to cause the receiver
                                                to re-enable automatically
#define SYS_CFG_AUTOACK         0x40000000UL    // Automatic Acknowledgement
                                                Enable
#define SYS_CFG_AACKPEND        0x80000000UL    // Automatic Acknowledgement
                                                Pending bit control

/****************************************************************************//
** @brief Bit definitions for register SYS_TIME
**/
#define SYS_TIME_ID             0x06            // System Time Counter (40-bit)
#define SYS_TIME_OFFSET         0x00
#define SYS_TIME_LEN            (5)             // Note 40 bit register

/****************************************************************************//
** @brief Bit definitions for register TX_FCTRL
**/
#define TX_FCTRL_ID             0x08            // Transmit Frame Control
#define TX_FCTRL_LEN            (5)             // Note 40 bit register
/* masks (low 32 bit) */
#define TX_FCTRL_TFLEN_MASK     0x0000007FUL    // bit mask to access Transmit
                                                Frame Length
#define TX_FCTRL_TFLE_MASK      0x00000380UL    // bit mask to access Transmit
                                                Frame Length Extension
#define TX_FCTRL_FLE_MASK       0x000003FFUL    // bit mask to access Frame
                                                Length field
#define TX_FCTRL_TXBR_MASK      0x00006000UL    // bit mask to access Transmit
                                                Bit Rate
#define TX_FCTRL_TXPRF_MASK     0x00030000UL    // bit mask to access Transmit
                                                Pulse Repetition Frequency
#define TX_FCTRL_TXPSR_MASK     0x000C0000UL    // bit mask to access Transmit
                                                Preamble Symbol Repetitions (PSR).
#define TX_FCTRL_PE_MASK        0x00300000UL    // bit mask to access Preamble
                                                Extension
#define TX_FCTRL_TXPSR_PE_MASK  0x003C0000UL    // bit mask to access Transmit
                                                Preamble Symbol Repetitions (PSR).
#define TX_FCTRL_SAFE_MASK_32   0xFFFFE3FFUL    // FSCTRL has fields which should
                                                always be writen zero
/* offset 0 */
/* offset 8 */
#define TX_FCTRL_TXBR_110k      0x00000000UL    // Transmit Bit Rate = 110k
#define TX_FCTRL_TXBR_850k      0x00002000UL    // Transmit Bit Rate = 850k
#define TX_FCTRL_TXBR_6M        0x00004000UL    // Transmit Bit Rate = 6.8M
#define TX_FCTRL_TXBR_SHFT      (13)            // shift to access Data Rate field
#define TX_FCTRL_TR             0x00008000UL    // Transmit Ranging enable
#define TX_FCTRL_TR_SHFT        (15)            // shift to access Ranging bit
/* offset 16 */
#define TX_FCTRL_TXPRF_SHFT     (16)            // shift to access Pulse
                                                Repetition Frequency field
#define TX_FCTRL_TXPRF_4M       0x00000000UL    // Transmit Pulse Repetition
                                                Frequency = 4 Mhz
```

```
#define TX_FCTRL_TXPRF_16M      0x00010000UL    // Transmit Pulse Repetition
                                                Frequency = 16 Mhz
#define TX_FCTRL_TXPRF_64M      0x00020000UL    // Transmit Pulse Repetition
                                                Frequency = 64 Mhz
#define TX_FCTRL_TXPSR_SHFT     (18)            // shift to access Preamble
                                                Symbol Repetitions field
#define TX_FCTRL_PE_SHFT        (20)            // shift to access Preamble length
                                                Extension to allow specification
                                                of non-standard values
#define TX_FCTRL_TXPSR_PE_16    0x00000000UL    // bit mask to access Preamble
                                                Extension = 16
#define TX_FCTRL_TXPSR_PE_64    0x00040000UL    // bit mask to access Preamble
                                                Extension = 64
#define TX_FCTRL_TXPSR_PE_128   0x00140000UL    // bit mask to access Preamble
                                                Extension = 128
#define TX_FCTRL_TXPSR_PE_256   0x00240000UL    // bit mask to access Preamble
                                                Extension = 256
#define TX_FCTRL_TXPSR_PE_512   0x00340000UL    // bit mask to access Preamble
                                                Extension = 512
#define TX_FCTRL_TXPSR_PE_1024  0x00080000UL    // bit mask to access Preamble
                                                Extension = 1024
#define TX_FCTRL_TXPSR_PE_1536  0x00180000UL    // bit mask to access Preamble
                                                Extension = 1536
#define TX_FCTRL_TXPSR_PE_2048  0x00280000UL    // bit mask to access Preamble
                                                Extension = 2048
#define TX_FCTRL_TXPSR_PE_4096  0x000C0000UL    // bit mask to access Preamble
                                                Extension = 4096
/* offset 22 */
#define TX_FCTRL_TXBOFFS_SHFT   (22)            // Shift to access transmit
                                                buffer index offset
#define TX_FCTRL_TXBOFFS_MASK   0xFFC00000UL    // bit mask to access Transmit
                                                buffer index offset 10-bit field
/* offset 32 */
#define TX_FCTRL_IFSDELAY_MASK  0xFF00000000ULL // bit mask to access Inter-Frame
                                                Spacing field

/******************************************************************************//
** @brief Bit definitions for register TX_BUFFER
**/
#define TX_BUFFER_ID            0x09            // Transmit Data Buffer

/******************************************************************************//
** @brief Bit definitions for register  DX_TIME
**/
#define DX_TIME_ID              0x0A            // Delayed Send or Receive Time
                                                (40-bit)

/******************************************************************************//
** @brief Bit definitions for register RX_FWTO
**/
#define RX_FWTO_ID              0x0C            // Receive Frame Wait Timeout
                                                Period
#define RX_FWTO_OFFSET          0x00
```

```
/*******************************************************************************//
** @brief Bit definitions for register SYS_CTRL
**/
#define SYS_CTRL_ID             0x0D            // System Control Register
#define SYS_CTRL_OFFSET         0x00
/* offset 0 */
#define SYS_CTRL_TXSTRT         0x00000002UL    // Start Transmitting Now
#define SYS_CTRL_TXDLYS         0x00000004UL    // Transmitter Delayed Sending
                                                (initiates sending when SYS_TIME
                                                == TXD_TIME
#define SYS_CTRL_TRXOFF         0x00000040UL    // Transceiver Off. Force
                                                Transciever OFF abort TX or RX
                                                immediately
#define SYS_CTRL_WAIT4RESP      0x00000080UL    // Wait for Response
/* offset 8 */
#define SYS_CTRL_RXENAB         0x00000100UL    // Enable Receiver Now
#define SYS_CTRL_RXDLYE         0x00000200UL    // Receiver Delayed Enable
/* offset 16 */
/* offset 24 */
#define SYS_CTRL_HRBT_OFFSET    (3)

/*******************************************************************************//
** @brief Bit definitions for register SYS_STATUS
**/
#define SYS_STATUS_ID           0x0F            // System event Status Register
#define SYS_STATUS_OFFSET       0x00
/* offset 0 */
#define SYS_STATUS_TXFRB        0x00000010UL    // Transmit Frame Begins
//#define SYS_STATUS_TXPRS        0x00000020UL    // Transmit Preamble Sent
#define SYS_STATUS_TXFRS        0x00000080UL    // Transmit Frame Sent: This is
                                                set when the transmitter has
                                                completed the sending of a frame
/* offset 8 */
#define SYS_STATUS_RXPHD        0x00000800UL    // Receiver PHY Header Detect
#define SYS_STATUS_RXPHE        0x00001000UL    // Receiver PHY Header Error
#define SYS_STATUS_RXDFR        0x00002000UL    // Receiver Data Frame Ready
#define SYS_STATUS_RXFCG        0x00004000UL    // Receiver FCS Good
#define SYS_STATUS_RXFCE        0x00008000UL    // Receiver FCS Error
/* offset 16 */
#define SYS_STATUS_RXRFSL       0x00010000UL    // Receiver Reed Solomon Frame
                                                Sync Loss
#define SYS_STATUS_RXRFTO       0x00020000UL    // Receive Frame Wait Timeout
#define SYS_STATUS_LDEERR       0x00040000UL    // Leading edge detection
                                                processing error
#define SYS_STATUS_RXOVRR       0x00100000UL    // Receiver Overrun
#define SYS_STATUS_RXPTO        0x00200000UL    // Preamble detection timeout
/* offset 24*/
#define SYS_STATUS_RXSFDTO      0x04000000UL    // Receive SFD timeout
#define SYS_STATUS_HPDWARN      0x08000000UL    // Half Period Delay Warning
#define SYS_STATUS_TXBERR       0x10000000UL    // Transmit Buffer Error
#define SYS_STATUS_AFFREJ       0x20000000UL    // Automatic Frame Filtering
                                                rejection
```

```
#define SYS_STATUS_HSRBP        0x40000000UL    // Host Side Receive Buffer
                                                Pointer
#define SYS_STATUS_ICRBP        0x80000000UL    // IC side Receive Buffer Pointer
                                                READ ONLY
/* offset 32 */
#define SYS_STATUS_RXPREJ       0x0200000000ULL // Receiver Preamble Rejection
#define SYS_STATUS_TXPUTE       0x0400000000ULL // Transmit power up time error
#define SYS_STATUS_TXERR        (0x0408)        // These bits are the 16 high
                                                bits of status register TXPUTE and
                                                HPDWARN flags
/* All RX events after a correct packet reception mask. */
#define SYS_STATUS_ALL_RX_GOOD (SYS_STATUS_RXDFR | SYS_STATUS_RXFCG | \
                                SYS_STATUS_RXPRD | SYS_STATUS_RXSFDD | \
                                SYS_STATUS_RXPHD | SYS_STATUS_LDEDONE)
/* All RX errors mask. */
#define SYS_STATUS_ALL_RX_ERR  (SYS_STATUS_RXPHE | SYS_STATUS_RXFCE | \
                                SYS_STATUS_RXRFSL | SYS_STATUS_RXSFDTO | \
                                SYS_STATUS_AFFREJ | SYS_STATUS_LDEERR)
/* User defined RX timeouts (frame wait timeout and preamble detect timeout)
 * mask. */
#define SYS_STATUS_ALL_RX_TO   (SYS_STATUS_RXRFTO | SYS_STATUS_RXPTO)
/* All TX events mask. */
#define SYS_STATUS_ALL_TX      (SYS_STATUS_AAT | SYS_STATUS_TXFRB | \
                                SYS_STATUS_TXPRS | SYS_STATUS_TXPHS | \
                                SYS_STATUS_TXFRS )

/***********************************************************************//
** @brief Bit definitions for register RX_FINFO
**/
#define RX_FINFO_ID             0x10            // RX Frame Information (in
                                                double buffer set)
#define RX_FINFO_OFFSET         0x00
/* mask and shift */
#define RX_FINFO_RXFLEN_MASK    0x0000007FUL    // Receive Frame Length (0 to 127)
#define RX_FINFO_RXFLE_MASK     0x00000380UL    // Receive Frame Length Extension
                                                (0 to 7)<<7
#define RX_FINFO_RXFL_MASK_1023 0x000003FFUL    // Receive Frame Length Extension
                                                (0 to 1023)
//#define RX_FINFO_RXNSPL_MASK    0x00001800UL    // Receive Non-Standard
                                                 Preamble Length
//#define RX_FINFO_RXPSR_MASK     0x000C0000UL    // RX Preamble Repetition. 00 =
                                                 16 symbols, 01 = 64 symbols, 10 =
                                                 1024 symbols, 11 = 4096 symbols
//#define RX_FINFO_RXPEL_MASK     0x000C1800UL    // Receive Preamble Length =
                                                 RXPSR+RXNSPL
//#define RX_FINFO_RXPEL_64       0x00040000UL    // Receive Preamble length = 64
//#define RX_FINFO_RXPEL_128      0x00040800UL    // Receive Preamble length = 128
//#define RX_FINFO_RXPEL_256      0x00041000UL    // Receive Preamble length = 256
//#define RX_FINFO_RXPEL_512      0x00041800UL    // Receive Preamble length = 512
//#define RX_FINFO_RXPEL_1024     0x00080000UL    // Receive Preamble length = 1024
//#define RX_FINFO_RXPEL_1536     0x00080800UL    // Receive Preamble length = 1536
//#define RX_FINFO_RXPEL_2048     0x00081000UL    // Receive Preamble length = 2048
//#define RX_FINFO_RXPEL_4096     0x000C0000UL    // Receive Preamble length = 4096
```

```
/****************************************************************************//
** @brief Bit definitions for register RX_BUFFER
**/
#define RX_BUFFER_ID            0x11            // Receive Data Buffer (in double
                                                buffer set)

/****************************************************************************//
** @brief Bit definitions for register RX_TIME
**/
#define RX_TIME_ID              0x15            // Receive Message Time of
                                                Arrival (in double buffer set)
#define RX_TIME_LLEN            (14)
#define RX_TIME_RX_STAMP_LEN    (5)             // read only 5 bytes (the
                                                adjusted timestamp (40:0))
/* mask and shift */
#define RX_TIME_RX_STAMP_OFFSET  (0)    // byte 0..4 40 bit Reports the fully
                                                adjusted time of reception.
#define RX_TIME_FP_INDEX_OFFSET  (5)    // byte 5..6 16 bit First path index.
#define RX_TIME_FP_AMPL1_OFFSET  (7)    // byte 7..8 16 bit First Path Amplitude
                                                point 1
#define RX_TIME_FP_RAWST_OFFSET  (9)    // byte 9..13 40 bit Raw Timestamp for
                                                the frame

/****************************************************************************//
** @brief Bit definitions for register
**/
#define TX_TIME_ID              0x17            // Transmit Message Time of Sending
#define TX_TIME_LLEN            (10)
#define TX_TIME_TX_STAMP_LEN    (5)             // 40-bits = 5 bytes
/* mask and shift */
#define TX_TIME_TX_STAMP_OFFSET (0)             // byte 0..4 40 bit Reports the
                                                fully adjusted time of
                                                transmission
#define TX_TIME_TX_RAWST_OFFSET (5)             // byte 5..9 40 bit Raw Timestamp
                                                for the frame

/****************************************************************************//
** @brief Bit definitions for register TX_ANTD
**/
#define TX_ANTD_ID              0x18            // 16-bit Delay from Transmit to
                                                Antenna
#define TX_ANTD_OFFSET          0x00

/****************************************************************************//
** @brief Bit definitions for register ACK_RESP_T
**/
/* Acknowledge (31:24 preamble symbol delay before auto ACK is sent) and respose
 * (19:0 - unit 1us) timer */
#define ACK_RESP_T_ID             0x1A          // Acknowledgement Time and
                                                Response Time
/* mask and shift */
```

```
#define ACK_RESP_T_MASK           0xFF0FFFFFUL  // Acknowledgement Time and
                                                Response access mask
#define ACK_RESP_T_W4R_TIM_OFFSET 0             // In bytes
#define ACK_RESP_T_W4R_TIM_MASK   0x000FFFFFUL  // Wait-for-Response turn-around
                                                Time 20 bit field
#define W4R_TIM_MASK             ACK_RESP_T_W4R_TIM_MASK
#define ACK_RESP_T_ACK_TIM_OFFSET 3             // In bytes
#define ACK_RESP_T_ACK_TIM_MASK   0xFF000000UL  // Auto-Acknowledgement turn-
                                                around Time
#define ACK_TIM_MASK              ACK_RESP_T_ACK_TIM_MASK

/****************************************************************************//
** @brief Bit definitions for register CHAN_CTRL
**/
#define CHAN_CTRL_ID             0x1F           // Channel Control
/* mask and shift */
#define CHAN_CTRL_MASK           0xFFFF00FFUL    // Channel Control Register
                                                access mask
#define CHAN_CTRL_TX_CHAN_MASK  0x0000000FUL    // Supported channels are 1, 2,
                                                3, 4, 5, and 7.
#define CHAN_CTRL_TX_CHAN_SHIFT (0)             // Bits 0..3        TX channel
                                                number 0-15 selection
#define CHAN_CTRL_RX_CHAN_MASK  0x000000F0UL
#define CHAN_CTRL_RX_CHAN_SHIFT (4)             // Bits 4..7        RX channel
                                                number 0-15 selection
#define CHAN_CTRL_RXFPRF_MASK   0x000C0000UL    // Bits 18..19      Specify (Force)
                                                RX Pulse Repetition Rate: 00 = 4
                                                MHz, 01 = 16 MHz, 10 = 64MHz.
#define CHAN_CTRL_RXFPRF_SHIFT  (18)
/* Specific RXFPRF configuration */
#define CHAN_CTRL_RXFPRF_4      0x00000000UL    // Specify (Force) RX Pulse
                                                Repetition Rate: 00 = 4 MHz, 01 =
                                                16 MHz, 10 = 64MHz.
#define CHAN_CTRL_RXFPRF_16     0x00040000UL    // Specify (Force) RX Pulse
                                                Repetition Rate: 00 = 4 MHz, 01 =
                                                16 MHz, 10 = 64MHz.
#define CHAN_CTRL_RXFPRF_64     0x00080000UL    // Specify (Force) RX Pulse
                                                Repetition Rate: 00 = 4 MHz, 01 =
                                                16 MHz, 10 = 64MHz.
#define CHAN_CTRL_TX_PCOD_MASK  0x07C00000UL    // Bits 22..26      TX Preamble
                                                Code selection, 1 to 24.
#define CHAN_CTRL_TX_PCOD_SHIFT (22)
#define CHAN_CTRL_RX_PCOD_MASK  0xF8000000UL    // Bits 27..31      RX Preamble
                                                Code selection, 1 to 24.
#define CHAN_CTRL_RX_PCOD_SHIFT (27)
/* offset 16 */
#define CHAN_CTRL_DWSFD         0x00020000UL    // Bit 17 This bit enables a non-
                                                standard DecaWave proprietary SFD
                                                sequence.
#define CHAN_CTRL_DWSFD_SHIFT   (17)
#define CHAN_CTRL_TNSSFD        0x00100000UL    // Bit 20 Non-standard SFD in the
                                                transmitter
#define CHAN_CTRL_TNSSFD_SHIFT  (20)
```

```
#define CHAN_CTRL_RNSSFD        0x00200000UL    // Bit 21 Non-standard SFD in the
                                                receiver
#define CHAN_CTRL_RNSSFD_SHIFT  (21)

/****************************************************************************//
** @brief Bit definitions for register USR_SFD
**        Please read User Manual: User defined SFD sequence
**/
#define USR_SFD_ID              0x21            // User-specified short/long
                                                TX/RX SFD sequences
#define DW_NS_SFD_LEN_110K      64              // Decawave non-standard SFD
                                                length for 110 kbps
#define DW_NS_SFD_LEN_850K      16              // Decawave non-standard SFD
                                                length for 850 kbps
#define DW_NS_SFD_LEN_6M8       8               // Decawave non-standard SFD
                                                length for 6.8 Mbps

/****************************************************************************//
** @brief Bit definitions for register AGC_CTRL
** Please take care to write to this register as doing so may cause the DW1000 to
** malfunction
**/
#define AGC_CTRL_ID             0x23            // Automatic Gain Control
                                                configuration
#define AGC_CFG_STS_ID          AGC_CTRL_ID
/* offset from AGC_CTRL_ID in bytes */
// Please take care not to write other values to this register as doing so may
   cause the DW1000 to malfunction
#define AGC_TUNE1_OFFSET        (0x04)
#define AGC_TUNE1_16M           0x8870
#define AGC_TUNE1_64M           0x889B
/* offset from AGC_CTRL_ID in bytes */
// Please take care not to write other values to this register as doing so may
   cause the DW1000 to malfunction
#define AGC_TUNE2_OFFSET        (0x0C)
#define AGC_TUNE2_VAL           0X2502A907UL

/****************************************************************************//
** @brief Bit definitions for register EXT_SYNC
**/
#define EXT_SYNC_ID             0x24            // External synchronisation control
/* offset from EXT_SYNC_ID in bytes */
#define EC_CTRL_OFFSET          (0x00)
#define EC_CTRL_PLLLCK          0x04            // PLL lock detect enable
#define EC_CTRL_OSTRM           0x00000800UL    // External timebase reset mode
                                                enable
#define EC_CTRL_WAIT_MASK       0x000007F8UL    // Wait counter used for external
                                                transmit synchronisation and
                                                external timebase reset
/* offset from EXT_SYNC_ID in bytes */
#define EC_RXTC_OFFSET          (0x04)
/* offset from EXT_SYNC_ID in bytes */
#define EC_GOLP                 (0x08)
```

```
/****************************************************************************//
** @brief Bit definitions for register  DRX_CONF
** Digital Receiver configuration block
**/
#define DRX_CONF_ID              0x27            // Digital Receiver configuration
/* offset from DRX_CONF_ID in bytes */
#define DRX_TUNE0b_OFFSET        (0x02)          // sub-register 0x02 is a 16-bit
                                                tuning register.
#define DRX_TUNE0b_110K_STD      0x000A
#define DRX_TUNE0b_110K_NSTD     0x0016
#define DRX_TUNE0b_850K_STD      0x0001
#define DRX_TUNE0b_850K_NSTD     0x0006
#define DRX_TUNE0b_6M8_STD       0x0001
#define DRX_TUNE0b_6M8_NSTD      0x0002
/* offset from DRX_CONF_ID in bytes */
#define DRX_TUNE1a_OFFSET        0x04            // 7.2.40.3 Sub-Register 0x27:04
                                                - DRX_TUNE1a
#define DRX_TUNE1a_PRF16         0x0087
#define DRX_TUNE1a_PRF64         0x008D
/* offset from DRX_CONF_ID in bytes */
#define DRX_TUNE1b_OFFSET        0x06            // 7.2.40.4 Sub-Register 0x27:06
                                                - DRX_TUNE1b
#define DRX_TUNE1b_110K          0x0064
#define DRX_TUNE1b_850K_6M8      0x0020
#define DRX_TUNE1b_6M8_PRE64     0x0010
/* offset from DRX_CONF_ID in bytes */
#define DRX_TUNE2_OFFSET         0x08            // 7.2.40.5 Sub-Register 0x27:08
                                                - DRX_TUNE2
#define DRX_TUNE2_PRF16_PAC8     0x311A002DUL
#define DRX_TUNE2_PRF16_PAC16    0x331A0052UL
#define DRX_TUNE2_PRF16_PAC32    0x351A009AUL
#define DRX_TUNE2_PRF16_PAC64    0x371A011DUL
#define DRX_TUNE2_PRF64_PAC8     0x313B006BUL
#define DRX_TUNE2_PRF64_PAC16    0x333B00BEUL
#define DRX_TUNE2_PRF64_PAC32    0x353B015EUL
#define DRX_TUNE2_PRF64_PAC64    0x373B0296UL
/* offset from DRX_CONF_ID in bytes */
/* WARNING: Please do NOT set DRX_SFDTOC to zero (disabling SFD detection
 * timeout) since this risks IC malfunction due to prolonged receiver activity in
 * the event of false preamble detection. */
#define DRX_SFDTOC_OFFSET        0x20            // 7.2.40.7 Sub-Register 0x27:20
                                                - DRX_SFDTOC
/* offset from DRX_CONF_ID in bytes */
#define DRX_PRETOC_OFFSET        0x24            // 7.2.40.9 Sub-Register 0x27:24
                                                - DRX_PRETOC
/* offset from DRX_CONF_ID in bytes */
#define DRX_TUNE4H_OFFSET        0x26            // 7.2.40.10 Sub-Register 0x27:26
                                                - DRX_TUNE4H
#define DRX_TUNE4H_PRE64         0x0010
#define DRX_TUNE4H_PRE128PLUS    0x0028
```

```
/****************************************************************************//
** @brief Bit definitions for register  RF_CONF
** Analog RF Configuration block
** Refer to section 7.2.41 Register file: 0x28 – Analog RF configuration block
**/
#define RF_CONF_ID              0x28            // Analog RF Configuration
/* offset from TX_CAL_ID in bytes */
#define RF_RXCTRLH_OFFSET       0x0B            // Analog RX Control Register
#define RF_RXCTRLH_NBW          0xD8            // RXCTRLH value for narrow
                                                bandwidth channels
#define RF_RXCTRLH_WBW          0xBC            // RXCTRLH value for wide
                                                bandwidth channels
/* offset from TX_CAL_ID in bytes */
#define RF_TXCTRL_OFFSET        0x0C            // Analog TX Control Register
#define RF_TXCTRL_CH1           0x00005C40UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL
#define RF_TXCTRL_CH2           0x00045CA0UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL
#define RF_TXCTRL_CH3           0x00086CC0UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL
#define RF_TXCTRL_CH4           0x00045C80UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL
#define RF_TXCTRL_CH5           0x001E3FE0UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL
#define RF_TXCTRL_CH7           0x001E7DE0UL    // 32-bit value to program to
                                                Sub-Register 0x28:0C – RF_TXCTRL

/****************************************************************************//
** @brief Bit definitions for register
** Refer to section 7.2.44 Register file: 0x2B – Frequency synthesiser control
** block
**/
#define FS_CTRL_ID              0x2B            // Frequency synthesiser control
                                                block
/* offset from FS_CTRL_ID in bytes */
#define FS_RES1_OFFSET          0x00            // reserved area. Please take
                                                care not to write to this area as
                                                doing so may cause the DW1000 to
                                                malfunction.
/* offset from FS_CTRL_ID in bytes */
#define FS_PLLCFG_OFFSET        0x07            // Frequency synthesiser – PLL
                                                configuration
#define FS_PLLCFG_CH1           0x09000407UL    // Operating Channel 1
#define FS_PLLCFG_CH2           0x08400508UL    // Operating Channel 2
#define FS_PLLCFG_CH3           0x08401009UL    // Operating Channel 3
#define FS_PLLCFG_CH4           FS_PLLCFG_CH2   // Operating Channel 4 (same as 2)
#define FS_PLLCFG_CH5           0x0800041DUL    // Operating Channel 5
#define FS_PLLCFG_CH7           FS_PLLCFG_CH5   // Operating Channel 7 (same as 5)
/* offset from FS_CTRL_ID in bytes */
#define FS_PLLTUNE_OFFSET       0x0B            // Frequency synthesiser – PLL
                                                Tuning
#define FS_PLLTUNE_CH1          0x1E            // Operating Channel 1
#define FS_PLLTUNE_CH2          0x26            // Operating Channel 2
```

```
#define FS_PLLTUNE_CH3          0x56            // Operating Channel 3
#define FS_PLLTUNE_CH4          FS_PLLTUNE_CH2  // Operating Channel 4 (same as 2)
#define FS_PLLTUNE_CH5          0xBE            // Operating Channel 5
#define FS_PLLTUNE_CH7          FS_PLLTUNE_CH5  // Operating Channel 7 (same as 5)
/* offset from FS_CTRL_ID in bytes */
#define FS_RES2_OFFSET          0x0C            // reserved area. Please take
                                                care not to write to this area as
                                                doing so may cause the DW1000 to
                                                malfunction.
/* offset from FS_CTRL_ID in bytes */
#define FS_XTALT_OFFSET         0x0E           // Frequency synthesiser - Crystal
                                                trim
#define FS_XTALT_MASK           0x1F           // Crystal Trim. Crystals may be
                                                trimmed using this register
                                                setting to tune out errors, see
                                                8.1 - IC Calibration - Crystal
                                                Oscillator Trim.
#define FS_XTALT_MIDRANGE       0x10
/* offset from FS_CTRL_ID in bytes */
#define FS_RES3_OFFSET          0x0F           // reserved area. Please take care
                                                not to write to this area as doing
                                                so may cause the DW1000 to
                                                malfunction.

/****************************************************************************//
** @brief Bit definitions for register
**/
#define AON_ID                  0x2C            // Always-On register set
/* offset from AON_ID in bytes */
#define AON_WCFG_OFFSET         0x00            // used to control what the DW1000
                                                IC does as it wakes up from low-
                                                power SLEEP or DEEPSLEEPstates.
#define AON_WCFG_ONW_RX         0x0002          // On Wake-up turn on the
                                                Receiver
#define AON_WCFG_ONW_LEUI       0x0008          // On Wake-up load the EUI from
                                                OTP memory into Register file:
                                                0x01 - Extended Unique Identifier.
#define AON_WCFG_ONW_LDC        0x0040          // On Wake-up load configurations
                                                from the AON memory into the host
                                                interface register set
#define AON_WCFG_ONW_L64P       0x0080          // On Wake-up load the Length64
                                                receiver operating parameter set
#define AON_WCFG_ONW_LLDE       0x0800          // On Wake-up load the LDE
                                                microcode.
#define AON_WCFG_ONW_LLDO       0x1000          // On Wake-up load the LDO tune
                                                value.
/* offset from AON_ID in bytes */
#define AON_CTRL_OFFSET         0x02            // The bits in this register in
                                                general cause direct activity
                                                within the AON block with respect
                                                to the stored AON memory
#define AON_CTRL_RESTORE        0x01            // When this bit is set the DW1000
                                                will copy the user configurations
```

```
                                                   from the AON memory to the host
                                                   interface register set.
#define AON_CTRL_SAVE           0x02               // When this bit is set the
                                                   DW1000 will copy the user
                                                   configurations from the host
                                                   interface register set into the
                                                   AON memory
#define AON_CTRL_UPL_CFG        0x04               // Upload the AON block
                                                   configurations to the AON
/* offset from AON_ID in bytes */
#define AON_RDAT_OFFSET         0x03               // AON Direct Access Read Data
                                                   Result
/* offset from AON_ID in bytes */
#define AON_ADDR_OFFSET         0x04               // AON Direct Access Address
/* offset from AON_ID in bytes */
#define AON_CFG0_OFFSET         0x06               // 32-bit configuration register
                                                   for the always on block.
/* offset from AON_ID in bytes */
#define AON_CFG1_OFFSET         0x0A

/****************************************************************************//
** @brief Bit definitions for register OTP_IF
** Refer to section 7.2.46 Register file: 0x2D - OTP Memory Interface
**/
#define OTP_IF_ID               0x2D               // One Time Programmable Memory
                                                   Interface
/* offset from OTP_IF_ID in bytes */
#define OTP_WDAT                0x00               // 32-bit register. The data
                                                   value to be programmed into an OTP
                                                   location
/* offset from OTP_IF_ID in bytes */
#define OTP_ADDR                0x04               // 16-bit register used to select
                                                   the address within the OTP memory
                                                   block
/* offset from OTP_IF_ID in bytes */
#define OTP_CTRL                0x06               // used to control the operation
                                                   of the OTP memory
#define OTP_CTRL_OTPRDEN        0x0001             // This bit forces the OTP into
                                                   manual read mode
#define OTP_CTRL_OTPREAD        0x0002             // This bit commands a read
                                                   operation from the address
                                                   specified in the OTP_ADDR register
#define OTP_CTRL_LDELOAD        0x8000             // This bit forces a load of LDE
                                                   microcode
#define OTP_CTRL_OTPPROG        0x0040             // Setting this bit will cause
                                                   the contents of OTP_WDAT to be
                                                   written to OTP_ADDR.
/* offset from OTP_IF_ID in bytes */
#define OTP_STAT                0x08
/* offset from OTP_IF_ID in bytes */
#define OTP_RDAT                0x0A               // 32-bit register. The data
                                                   value read from an OTP location
                                                   will appear here
```

```
/* offset from OTP_IF_ID in bytes */
#define OTP_SRDAT               0x0E            // 32-bit register. The data
                                                value stored in the OTP SR (0x400)
                                                location will appear here after
                                                power up
/* offset from OTP_IF_ID in bytes */
#define OTP_SF                  0x12            // 8-bit special function
                                                register used to select and load
                                                special receiver operational
                                                parameter
#define OTP_SF_OPS_KICK         0x01            // This bit when set initiates a
                                                load of the operating parameter
                                                set selected by the OPS_SEL
#define OTP_SF_LDO_KICK         0x02            // This bit when set initiates a
                                                load of the LDO tune code
#define OTP_SF_OPS_SEL_L64      0x00            // Operating parameter set
                                                selection: Length64
#define OTP_SF_OPS_SEL_TIGHT    0x40            // Operating parameter set
                                                selection: Tight

/****************************************************************************//
** @brief Bit definitions for register LDE_IF
** Refer to section 7.2.47 Register file: 0x2E - Leading Edge Detection Interface
** PLEASE NOTE: Other areas within the address space of Register file: 0x2E -
** Leading Edge Detection Interface are reserved. To ensure proper operation of
** the LDE algorithm (i.e. to avoid loss of performance or a malfunction), care
** must be taken not to write to any byte locations other than those defined in
** the sub-sections below.
**/
#define LDE_IF_ID               0x2E            // Leading edge detection control
                                                block
/* offset from LDE_IF_ID in bytes */
#define LDE_THRESH_OFFSET       0x0000          // 16-bit status register
                                                reporting the threshold that was
                                                used to find the first path
/* offset from LDE_IF_ID in bytes */
#define LDE_CFG1_OFFSET         0x0806          // 8-bit configuration register
/* offset from LDE_IF_ID in bytes */
#define LDE_PPINDX_OFFSET       0x1000          // reporting the position within
                                                the accumulator that the LDE
                                                algorithm has determined to
                                                contain the maximum
/* offset from LDE_IF_ID in bytes */
#define LDE_PPAMPL_OFFSET       0x1002          // reporting the magnitude of the
                                                peak signal seen in the
                                                accumulator data memory
/* offset from LDE_IF_ID in bytes */
#define LDE_RXANTD_OFFSET       0x1804          // 16-bit configuration register
                                                for setting the receive antenna
                                                delay
/* offset from LDE_IF_ID in bytes */
#define LDE_CFG2_OFFSET         0x1806          // 16-bit LDE configuration
                                                tuning register
```

```
/* offset from LDE_IF_ID in bytes */
#define LDE_REPC_OFFSET         0x2804          // 16-bit configuration register
                                                for setting the replica avoidance
                                                coefficient
#define LDE_REPC_PCODE_1        0x5998
#define LDE_REPC_PCODE_2        0x5998
#define LDE_REPC_PCODE_3        0x51EA
#define LDE_REPC_PCODE_4        0x428E
#define LDE_REPC_PCODE_5        0x451E
#define LDE_REPC_PCODE_6        0x2E14
#define LDE_REPC_PCODE_7        0x8000
#define LDE_REPC_PCODE_8        0x51EA
#define LDE_REPC_PCODE_9        0x28F4
#define LDE_REPC_PCODE_10       0x3332
#define LDE_REPC_PCODE_11       0x3AE0
#define LDE_REPC_PCODE_12       0x3D70
#define LDE_REPC_PCODE_13       0x3AE0
#define LDE_REPC_PCODE_14       0x35C2
#define LDE_REPC_PCODE_15       0x2B84
#define LDE_REPC_PCODE_16       0x35C2
#define LDE_REPC_PCODE_17       0x3332
#define LDE_REPC_PCODE_18       0x35C2
#define LDE_REPC_PCODE_19       0x35C2
#define LDE_REPC_PCODE_20       0x47AE
#define LDE_REPC_PCODE_21       0x3AE0
#define LDE_REPC_PCODE_22       0x3850
#define LDE_REPC_PCODE_23       0x30A2
#define LDE_REPC_PCODE_24       0x3850

/****************************************************************************//
** @brief Bit definitions for register DIG_DIAG
** Digital Diagnostics interface.
** It contains a number of sub-registers that give diagnostics information.
**/
#define DIG_DIAG_ID             0x2F            // Digital Diagnostics Interface

/****************************************************************************//
** @brief Bit definitions for register PMSC
**/
#define PMSC_ID                     0x36        // Power Management System
                                                Control Block
//#define PMSC_LEN                   (48)
#define PMSC_CTRL0_OFFSET           0x00
#define PMSC_CTRL0_SOFTRESET_OFFSET 3           // In bytes
#define PMSC_CTRL0_RESET_ALL        0x00        // Assuming only 4th byte of the
                                                register is read
#define PMSC_CTRL0_RESET_RX         0xE0        // Assuming only 4th byte of the
                                                register is read
#define PMSC_CTRL0_RESET_CLEAR      0xF0        // Assuming only 4th byte of the
                                                register is read
/* offset from PMSC_ID in bytes */
#define PMSC_CTRL1_OFFSET           0x04
```

```
#define PMSC_CTRL1_PKTSEQ_DISABLE   0x00        // writing this to PMSC CONTROL 1
                                                register (bits 10-3) disables PMSC
                                                control of analog RF subsystems
#define PMSC_CTRL1_PKTSEQ_ENABLE    0xE7        // writing this to PMSC CONTROL 1
                                                register (bits 10-3) enables PMSC
                                                control of analog RF subsystems


#ifdef __cplusplus
}
#endif

#endif
```

---

deca_spi.h

```
/*! ----------------------------------------------------------------------------
 * @file deca_spi.h
 * @brief SPI access functions
 * Copyright 2013 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 14.2.17
 */

#ifndef _DECA_SPI_H_
#define _DECA_SPI_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "deca_types.h"
#include <xc.h>

#define DECA_MAX_SPI_HEADER_LENGTH    (3)       // max number of bytes in header
                                                (for formating & sizing)
#define SPI_ON                       SPI4CONbits.ON
#define SPI_BUSY                     !SPI4STATbits.SPIRBF
#define SSLAT                        LATBbits.LATB8

/*! ----------------------------------------------------------------------------
 * @fn openspi()
 * @originates deca_spi.c
 */
int openspi(void);

/*! ----------------------------------------------------------------------------
 * @fn closespi()
 * @originates deca_spi.c
 */
int closespi(void);
/*! ----------------------------------------------------------------------------
 * @fn spi_transaction()
 * @originates deca_spi.c
 */
unsigned char spi_transaction(unsigned char data);


#ifdef __cplusplus
}
#endif

#endif /* _DECA_SPI_H_ */
```

---

deca_types.h

```
/*! ----------------------------------------------------------------------------
 * @file deca_types.h
 * @brief Decawave general type definitions
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#ifndef _DECA_TYPES_H_
#define _DECA_TYPES_H_

#ifdef __cplusplus
extern "C" {
#endif


#ifndef uint8
#ifndef _DECA_UINT8_
#define _DECA_UINT8_
typedef unsigned char uint8;
#endif
#endif

#ifndef uint16
#ifndef _DECA_UINT16_
#define _DECA_UINT16_
typedef unsigned short uint16;
#endif
#endif

#ifndef uint32
#ifndef _DECA_UINT32_
#define _DECA_UINT32_
typedef unsigned long uint32;
#endif
#endif

#ifndef int8
#ifndef _DECA_INT8_
#define _DECA_INT8_
typedef signed char int8;
#endif
#endif

#ifndef int16
#ifndef _DECA_INT16_
#define _DECA_INT16_
typedef signed short int16;
#endif
#endif

#ifndef int32
#ifndef _DECA_INT32_
#define _DECA_INT32_
typedef signed long int32;
#endif
#endif

#ifndef NULL
#define NULL ((void *)0UL)
#endif


#ifdef __cplusplus
}
#endif

#endif /* DECA_TYPES_H_ */
```

---

deca_version.h

```
/*! ----------------------------------------------------------------------------
 * @file deca_version.h
 * @brief Defines the version info for the DW1000 device driver including its API
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#ifndef _DECA_VERSION_H_
#define _DECA_VERSION_H_

/* The DW1000 device driver is separately version numbered to any version the
 * application using it may have */

/* Two symbols are defined here: one hexadecimal value and one string that
 * includes the hex bytes. Both should be updated together in a consistent way
 * when the software is being modified. */

/* The format of the hex version is 0xAABBCC and the string ends with AA.BB.CC,
 * where...
 * Quantity CC is updated for minor changes/bug fixes that should not need user
 * code changes
 * Quantity BB is updated for changes/bug fixes that may need user code changes
 * Quantity AA is updated for major changes that will need user code changes */

#define DW1000_DRIVER_VERSION             0x040002
#define DW1000_DEVICE_DRIVER_VER_STRING   "DW1000 Device Driver Version 04.00.02"

#endif
```

---

functions17.h

```
/*
 * @file functions17.h
 * @author Katherine Sanders
 * @brief Definitions required to run RFPS board
 *
 * edited 25.4.17
 */

#ifndef FUNCTIONS17_H
#define    FUNCTIONS17_H

#ifdef     __cplusplus
extern "C" {
#endif

#include "deca_types.h"
#include <xc.h>
#include <stdlib.h>
#include <string.h>
//#include <stdio.h>


typedef unsigned long long uint64;

/*Inputs for the DIP switches - as of now only 1, 2, 3, and 8 are being used */
#define DIP1 PORTGbits.RG6
#define DIP2 PORTEbits.RE7
#define DIP3 PORTEbits.RE6
#define DIP4 PORTEbits.RE5
#define DIP5 PORTEbits.RE4
#define DIP6 PORTEbits.RE3
#define DIP7 PORTEbits.RE2
#define DIP8 PORTEbits.RE1

/* For toggle bits */
#define TRIS_D1                    TRISDbits.TRISD1
#define toggle_d1                  LATDbits.LATD1=1; _nop(); LATDbits.LATD1=0;
/* Example application name and version to display screen. */
#define APP_NAME                   "GET DISTANCE"
/* UWB microsecond (uus) to device time unit (dtu, around 15.65 ps) conversion
 * factor. 1 uus = 512 / 499.2 µs and 1 µs = 499.2 * 128 dtu. */
#define UUS_TO_DWT_TIME            65536
/* Inter-ranging delay period, in milliseconds. */
```

```
#define RNG_DELAY_MS                 50
/* fail handling */
#define ACCEPTABLE_FAILS             10
/* Default antenna delay values for 64 MHz PRF. */
#define TX_ANT_DLY                   16436        //original is 16436, but we used
                                                 16620 as well
#define RX_ANT_DLY                   16436        //original is 16436, but we used
                                                 16620 as well
/* Length of the common part of the message (up to and including the function
 * code). */
#define ALL_MSG_COMMON_LEN           10
/* Indexes to access some of the fields in the frames defined above. */
#define ALL_MSG_SN_IDX               2
#define RESP_MSG_POLL_RX_TS_IDX      10
#define RESP_MSG_RESP_TX_TS_IDX      14
#define RESP_MSG_TS_LEN              4
/* Delay between frames, in UWB microseconds. */
#define POLL_TX_TO_RESP_RX_DLY_UUS 140
#define POLL_RX_TO_RESP_TX_DLY_UUS 700            //original is 330
/* Receive response timeout. */
#define RESP_RX_TIMEOUT_UUS          1250         //original is210
/* Speed of light in air, in meters per second. */
#define SPEED_OF_LIGHT               299702547
/* Conversion factor from meters to inches */
#define METERS_TO_INCHES             39.37008
/* Buffer to store received response message.
 * Its size is adjusted to longest frame that this code is supposed to handle. */
#define RX_BUF_LEN                   20           //initializer
#define RX_BUFFER_LEN                12           //responder


/*! ----------------------------------------------------------------------------
 * @fn getu_3()
 * @originates functions17.c
 */
unsigned char getu_3(void); // getu (receive char) for UART3

/*! ----------------------------------------------------------------------------
 * @fn putu_3()
 * @originates functions17.c
 */
void putu_3(char val); // putu (send char) for UART3

/*! ----------------------------------------------------------------------------
 * @fn serial_3_init()
 * @originates functions17.c
 */
void serial_3_init(unsigned long rate);

/*! ----------------------------------------------------------------------------
 * @fn configDIP()
 * @originates functions17.c
 */
```

```
void configDIP(void);

/*! ------------------------------------------------------------------------------
 * @fn print_string()
 * @originates functions17.c
 */
void print_string(char *string);

/*! ------------------------------------------------------------------------------
 * @fn print_string2()
 * @originates functions17.c
 */
void print_string2(char *string); //print_string() for UART2

/*! ------------------------------------------------------------------------------
 * @fn print_string3()
 * @originates functions17.c
 */
void print_string3(char *string); //put_string() for UART3

/*! ------------------------------------------------------------------------------
 * @fn determine_parameters()
 * @originates functions17.c
 */
uint8 determine_parameters(uint8 switch1, uint8 switch2, uint8 switch3);

/*! ------------------------------------------------------------------------------
 * @fn initializer()
 * @originates initiator.c
 */
void initializer(int enableWifi);

/*! ------------------------------------------------------------------------------
 * @fn responder()
 * @originates responder.c
 */
void responder(uint8 num, int enableWifi);


#ifdef     __cplusplus
}
#endif

#endif     /* FUNCTIONS17_H */
```

lcd.h

```
/*! -----------------------------------------------------------------------------
 * @file lcd.h
 * @brief EVB1000 LCD screen access functions
 * Copyright 2015 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 3.3.17
 */

#ifndef _LCD_H_
#define _LCD_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "SDlib16.h"

/*! ------------------------------------------------------------------------------
 * @fn writetoLCD()
 * @originates lcd.c
 */
void writetoLCD
(
    int       bodylength,
    int        rs_enable,
    const char *bodyBuffer
);

/*! ------------------------------------------------------------------------------
 * @fn printHEX()
 * @originates lcd.c
 */
void printHEX(unsigned long num);

/*! ------------------------------------------------------------------------------
 * @fn lcd_display_str()
 * @originates lcd.c
 */
void lcd_display_str(const char *string);


#ifdef __cplusplus
}
#endif

#endif /* _LCD_H_ */
```

---

magnetometer.h

```
/*
 * @file magnetometer.h
 * @author Stephen McAndrew
 *         Eddie Hunckler
 * @brief Definitions required to run and get values from magnetometer (MAG3110)
 *        on RFPS board
 *
 * edited 26.4.17
 */
```

```c
#ifndef _MAGNETOMETER_H    // Guard against multiple inclusion
#define _MAGNETOMETER_H


#include <xc.h>
#include <sys/attribs.h>


#define MEASUREMENT_AVAILABLE PORTDbits.RD11
#define MEASUREMENT_PIN       TRISDbits.TRISD11

#define ACK                   0x01
#define NACK                  0x00

#define DR_STATUS             0x00
#define OUT_X_MSB             0x01
#define OUT_X_LSB             0x02
#define OUT_Y_MSB             0x03
#define OUT_Y_LSB             0x04
#define OUT_Z_MSB             0x05
#define OUT_Z_LSB             0x06
#define WHO_AM_I              0x07
#define SYSMOD                0x08
#define OFF_X_MSB             0x09
#define OFF_X_LSB             0x0A
#define OFF_Y_MSB             0x0B
#define OFF_Y_LSB             0x0C
#define OFF_Z_MSB             0x0D
#define OFF_Z_LSB             0x0E
#define DIE_TEMP              0x0F
#define CTRL_REG1             0x10
#define CTRL_REG2             0x11

#define READ_DEVICE           0x1D
#define WRITE_DEVICE          0x1C

/* Calibration Values */
#define XOFFSET               -6180
#define YOFFSET               6000
#define ZOFFSET               4000

/*! ------------------------------------------------------------------------------
 * @fn I2C_init()
 * @originates magnetometer.c
 */
unsigned char I2C_init(unsigned long rate);

/*! ------------------------------------------------------------------------------
 * @fn I2C_start()
 * @originates magnetometer.c
 */
unsigned char I2C_start(void);
```

```c
/*! -----------------------------------------------------------------------------
 * @fn I2C_stop()
 * @originates magnetometer.c
 */
unsigned char I2C_stop(void);

/*! -----------------------------------------------------------------------------
 * @fn I2C_restart()
 * @originates magnetometer.c
 */
unsigned char I2C_restart(void);

/*! -----------------------------------------------------------------------------
 * @fn I2C_write()
 * @originates magnetometer.c
 */
char I2C_write(char data);

/*! -----------------------------------------------------------------------------
 * @fn I2C_read()
 * @originates magnetometer.c
 */
unsigned char I2C_read(unsigned char ack);

/*! -----------------------------------------------------------------------------
 * @fn MAG3110_init()
 * @originates magnetometer.c
 */
void MAG3110_init(void);

/*! -----------------------------------------------------------------------------
 * @fn getX()
 * @originates magnetometer.c
 */
int getX(void);

/*! -----------------------------------------------------------------------------
 * @fn getY()
 * @originates magnetometer.c
 */
int getY(void);

/*! -----------------------------------------------------------------------------
 * @fn getZ()
 * @originates magnetometer.c
 */
int getZ(void);


unsigned int MSB_x;
unsigned int LSB_x;
unsigned char MSB_y;
unsigned char LSB_y;
unsigned char MSB_z;
unsigned char LSB_z;

unsigned int x_data;
unsigned int y_data;
unsigned int z_data;

int x_data_int;
int y_data_int;
int z_data_int;


#endif /* _MAGNETOMETER_H */
```

---

port.h

```
/*! ----------------------------------------------------------------------------
 * @file port.h
 * @brief HW specific definitions and functions for portability
 * Copyright 2013 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 24.4.17
 */

#include <stdint.h>


#ifndef PORT_H_
#define PORT_H_

#ifdef __cplusplus
extern "C" {
#endif


/* Define the wanted value of CLOCKS_PER_SEC to create a millisecond tick timer. */
#undef CLOCKS_PER_SEC
#define CLOCKS_PER_SEC       1000

//void printf2(const char *format, ...);

typedef enum
{
    LED_PC6,
    LED_PC7,
    LED_PC8,
    LED_PC9,
    LED_ALL,
    LEDn
} led_t;
#define RESETTRIS            TRISBbits.TRISB10
#define RESETLAT             LATBbits.LATB10
#define SSLAT                LATBbits.LATB8

/*! ------------------------------------------------------------------------------
 * @fn peripherals_init()
 * @originates port.c
 */
void peripherals_init (void);

/*! ------------------------------------------------------------------------------
 * @fn spi_set_rate_low()
 * @originates port.c
 */
void spi_set_rate_low (void);

/*! ------------------------------------------------------------------------------
 * @fn spi_set_rate_high()
 * @originates port.c
 */
void spi_set_rate_high (void);

/*! ------------------------------------------------------------------------------
 * @fn SPI_Configuration()
 * @originates port.c
 */
int SPI_Configuration(void);

/*! ------------------------------------------------------------------------------
 * @fn reset_DW1000()
 * @originates port.c
 */
void reset_DW1000(void);


#ifdef __cplusplus
}
#endif

#endif /* PORT_H_ */
```

SDLib16.h

```
/*
 * @file SDlib16.h
 * @author Mike Schafer
 * @brief Standard functions for PIC32
 *        2016 board with 8 MHz resonator
 *
 * edited 22.4.17
 */

#ifndef _SDLIB16_H_
```

```
#define _SDLIB16_H_

#include <xc.h>

/******************************************************************************
 *  2016 board version
 * header file for the support routines for the kit board
 * At present, included are:
 * Delay routines
 * serial (spi) LCD (SPI2 with D7 as chip select
 * serial port routines (UART1)
 * switcher to allow stdout to go to LCD or serial port
 */

/* LCD Function prototypes
 * specific to spi display */
void LCD_init(void);
void LCD_char(char val);
void LCD_display_on(void);
void LCD_display_off(void);
void LCD_clear(void);
void LCD_backlight(char val);
void LCD_contrast(char val);
void LCD_setpos(char row, char col);

/* simple input routines */
unsigned int getdec(void);
unsigned int gethex(void);
signed int getint(void);

/* Serial I/O via USART*/
unsigned char getu(void);                      // get char
void putu(unsigned char val);                  // put char
void serial_init(unsigned long rate);
#define u1_buff_full        U1STAbits.UTXBF    // int flag for tx buff
#define u1_data_avail       U1STAbits.URXDA    // int flag for rx buff
unsigned char getu2(void);                     // get char
void putu2(unsigned char val);                 // put char
void serial2_init(unsigned long rate);
#define u2_buff_full        U2STAbits.UTXBF    // int flag for tx buff
#define u2_data_avail       U2STAbits.URXDA    // int flag for rx buff

void set_output_device(unsigned char device);

/* delay routines*/
void set_sys_clock(unsigned long val);
unsigned long get_sys_clock(void);
void set_pb_clock(unsigned long val);
unsigned long get_pb_clock(void);
void delay_ms(unsigned long val);
void delay_us(unsigned long val);
```

```
#endif //SDLIB16_H
```

---

sleep.h

```
/*! ----------------------------------------------------------------------------
 * @file sleep.h
 * @brief platform dependent sleep implementation
 * Copyright 2015 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#ifndef _SLEEP_H_
#define _SLEEP_H_

#ifdef __cplusplus
extern "C" {
#endif

/*! ------------------------------------------------------------------------------
 * @fn sleep_ms()
 * @originates deca_sleep.c
 */
void sleep_ms(unsigned int time_ms);

#ifdef __cplusplus
}
#endif

#endif /* _SLEEP_H_ */
```

2. Source Files

deca_device.c

```
/*! ----------------------------------------------------------------------------
 * @file deca_device.c
 * @brief Decawave device configuration and control functions
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 24.4.17
 */

#include "deca_device_api.h"
#include "deca_param_types.h"
#include "deca_regs.h"
#include "deca_types.h"
#include "lcd.h"
#include <assert.h>

/* Defines for enable_clocks function */
#define FORCE_SYS_XTI      0
#define ENABLE_ALL_SEQ     1
#define FORCE_SYS_PLL      2
#define READ_ACC_ON        7
#define READ_ACC_OFF       8
#define FORCE_OTP_ON       11
#define FORCE_OTP_OFF      12
#define FORCE_TX_PLL       13
#define FORCE_LDE          14

/* Defines for ACK request bitmask in DATA and MAC COMMAND frame control (first
 * byte) - Used to detect AAT bit wrongly set. */
#define FCTRL_ACK_REQ_MASK 0x20
/* Frame control maximum length in bytes. */
#define FCTRL_LEN_MAX      2


/* Enable and Configure specified clocks */
void _dwt_enableclocks(int clocks) ;
/* Configure the ucode (FP algorithm) parameters */
void _dwt_configlde(int prf);
/* Load ucode from OTP/ROM */
void _dwt_loaducodefromrom(void);
/* Read non-volatile memory */
uint32 _dwt_otpread(uint32 address);
/* Program the non-volatile memory */
uint32 _dwt_otpprogword32(uint32 data, uint16 address);
/* Upload the device configuration into always on memory */

// ----------------------------------------------------------------------------
// Structure to hold device data
typedef struct
{
    uint32      partID;             // IC Part ID - read during initialization
    uint32      lotID;              // IC Lot ID - read during initialization
    uint8       longFrames;         // Flag in non-standard long frame mode
    uint8       otprev;             // OTP revision number (read during
                                    initialization)
    uint32      txFCTRL;            // Keep TX_FCTRL register config
    uint8       init_xtrim;         // initial XTAL trim value read from OTP (or
                                    defaulted to mid-range if OTP not programmed)
    uint8       dblbuffon;          // Double RX buffer mode flag
    uint32      sysCFGreg;          // Local copy of system config register
    uint16      sleep_mode;         // Used for automatic reloading of LDO tune
                                    and microcode at wake-up
    uint8       wait4resp;          // wait4response was set with last TX start
                                    command
    dwt_cb_data_t cbData;           // Callback data structure
    dwt_cb_t    cbTxDone;           // Callback for TX confirmation event
    dwt_cb_t    cbRxOk;             // Callback for RX good frame event
    dwt_cb_t    cbRxTo;             // Callback for RX timeout events
    dwt_cb_t    cbRxErr;            // Callback for RX error events
} dwt_local_data_t;

static dwt_local_data_t dw1000local; // Static local device data


/*! ------------------------------------------------------------------------------------
 * @fn dwt_initialise()
 * @brief This function initiates communications with the DW1000 transceiver
 *        and reads its DEV_ID register (address 0x00) to verify the IC is one
 *        supported by this software (e.g. DW1000 32-bit device ID value is
 *        0xDECA0130). Then it does any initial once only device configurations
 *        needed for use and initializes as necessary any static data items
 *        belonging to this low-level driver.
 * NOTES: 1. This function needs to be run before dwt_configuresleep, also the
 *           SPI frequency has to be < 3MHz
 *        2. It also reads and applies LDO tune and crystal trim values from OTP
 *           memory
 * @param config -  specifies what configuration to load
 *                  DWT_LOADUCODE     0x1 - load the LDE microcode from ROM –
 *                                          enabled accurate RX timestamp
 *                  DWT_LOADNONE      0x0 - do not load any values from OTP memory
 * @returns DWT_SUCCESS - success
 *          DWT_ERROR   - error
 */
/* OTP addresses definitions */
#define LDOTUNE_ADDRESS    (0x04)
#define PARTID_ADDRESS     (0x06)
#define LOTID_ADDRESS      (0x07)
#define VBAT_ADDRESS       (0x08)
#define VTEMP_ADDRESS      (0x09)
#define XTRIM_ADDRESS      (0x1E)

int dwt_initialise(uint16 config)
{
```

```
uint16 otp_addr = 0;
uint32 ldo_tune = 0;

dw1000local.dblbuffon = 0; // Double buffer mode off by default
dw1000local.wait4resp = 0;
dw1000local.sleep_mode = 0;
dw1000local.cbTxDone = NULL;
dw1000local.cbRxOk = NULL;
dw1000local.cbRxTo = NULL;
dw1000local.cbRxErr = NULL;
/* Read and validate device ID. Return -1 if not recognised */
if (DWT_DEVICE_ID != dwt_readdevid()) // MP IC ONLY (i.e. DW1000) for this code
{
    return DWT_ERROR;
}
/* Make sure the device is completely reset before starting initialisation */
dwt_softreset();
_dwt_enableclocks(FORCE_SYS_XTI); // NOTE: set system clock to XTI -
                                  //this is necessary to make sure the values
                                  read by _dwt_otpread are reliable
/* Configure the CPLL lock detect */
dwt_write8bitoffsetreg(EXT_SYNC_ID, EC_CTRL_OFFSET, EC_CTRL_PLLLCK);
/* Read OTP revision number */
otp_addr = _dwt_otpread(XTRIM_ADDRESS) & 0xffff; // Read 32 bit value, XTAL
                                                 trim val is in low octet-0
                                                 (5 bits)
dw1000local.otprev = (otp_addr >> 8) & 0xff;     // OTP revision is next byte
/* Load LDO tune from OTP and kick it if there is a value programmed. */
ldo_tune = _dwt_otpread(LDOTUNE_ADDRESS);
if((ldo_tune & 0xFF) != 0)
{
    /* Kick LDO tune */
    dwt_write8bitoffsetreg(OTP_IF_ID, OTP_SF, OTP_SF_LDO_KICK); // Set load
                                                                LDO kick bit
    dw1000local.sleep_mode |= AON_WCFG_ONW_LLDO; // LDO tune must be kicked
                                                 at wake-up
}
/* Load Part and Lot ID from OTP */
dw1000local.partID = _dwt_otpread(PARTID_ADDRESS);
dw1000local.lotID = _dwt_otpread(LOTID_ADDRESS);
/* XTAL trim value is set in OTP for DW1000 module and EVK/TREK boards but
 * that might not be the case in a custom design */
dw1000local.init_xtrim = otp_addr & 0x1F;
if (!dw1000local.init_xtrim) // A 0 means the crystal hasn't been trimmed
{
    dw1000local.init_xtrim = FS_XTALT_MIDRANGE; // Set to mid-range if no
                                                calibration value inside
}
/* Configure XTAL trim */
dwt_setxtaltrim(dw1000local.init_xtrim);
/* Load leading edge detect code */
if(config & DWT_LOADUCODE)
{
        _dwt_loaducodefromrom();
        dw1000local.sleep_mode |= AON_WCFG_ONW_LLDE; // microcode must be loaded
                                                       at wake-up
    }
    else /* Should disable the LDERUN enable bit in 0x36, 0x4 */
    {
        uint16 rega = dwt_read16bitoffsetreg(PMSC_ID, PMSC_CTRL1_OFFSET+1);
        rega &= 0xFDFF; // Clear LDERUN bit
        dwt_write16bitoffsetreg(PMSC_ID, PMSC_CTRL1_OFFSET+1, rega);
    }
    _dwt_enableclocks(ENABLE_ALL_SEQ); // Enable clocks for sequencing
    /* The 3 bits in AON CFG1 register must be cleared to ensure proper operation
     * of the DW1000 in DEEPSLEEP mode. */
    dwt_write8bitoffsetreg(AON_ID, AON_CFG1_OFFSET, 0x00);
    /* Read system register and store local copy */
    dw1000local.sysCFGreg = dwt_read32bitreg(SYS_CFG_ID); // Read sysconfig
                                                            register

    return DWT_SUCCESS ;
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readdevid()
 * @brief This is used to return the read device type and revision information of
 * the DW1000 device (MP part is 0xDECA0130)
 * @param none
 * @returns the read value which for DW1000 is 0xDECA0130
 */
uint32 dwt_readdevid(void)
{
    return dwt_read32bitoffsetreg(DEV_ID_ID,0);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_configure()
 * @brief This function provides the main API for the configuration of the DW1000
 *        and this low-level driver. The input is a pointer to the data structure
 *        of type dwt_config_t that holds all the configurable items.
 *        The dwt_config_t structure shows which ones are supported
 * @param config - pointer to the configuration structure, which contains the
 *                 device configuration data.
 * @return none
 */
void dwt_configure(dwt_config_t *config)
{
    uint8 nsSfd_result  = 0;
    uint8 useDWnsSFD = 0;
    uint8 chan = config->chan;
    uint32 regval;
    uint16 reg16 = lde_replicaCoeff[config->rxCode];
    uint8 prfIndex = config->prf - DWT_PRF_16M;
    uint8 bw = ((chan == 4) || (chan == 7)) ? 1: 0; // Select wide or narrow band
```

```
#ifdef DWT_API_ERROR_CHECK
    assert(config->dataRate <= DWT_BR_6M8);
    assert(config->rxPAC <= DWT_PAC64);
    assert((chan >= 1) && (chan <= 7) && (chan != 6));
    assert(((config->prf == DWT_PRF_64M) && (config->txCode >= 9) &&
            (config->txCode <= 24)) || ((config->prf == DWT_PRF_16M) &&
            (config->txCode >= 1) && (config->txCode <= 8)));
    assert(((config->prf == DWT_PRF_64M) && (config->rxCode >= 9) &&
            (config->rxCode <= 24)) || ((config->prf == DWT_PRF_16M) &&
            (config->rxCode >= 1) && (config->rxCode <= 8)));
    assert((config->txPreambLength == DWT_PLEN_64) ||
           (config->txPreambLength == DWT_PLEN_128) ||
           (config->txPreambLength == DWT_PLEN_256) ||
           (config->txPreambLength == DWT_PLEN_512) ||
           (config->txPreambLength == DWT_PLEN_1024) ||
           (config->txPreambLength == DWT_PLEN_1536) ||
           (config->txPreambLength == DWT_PLEN_2048) ||
           (config->txPreambLength == DWT_PLEN_4096));
    assert((config->phrMode == DWT_PHRMODE_STD) ||
           (config->phrMode == DWT_PHRMODE_EXT));
#endif

    /* For 110 kbps we need a special setup */
    if(DWT_BR_110K == config->dataRate)
    {
        dw1000local.sysCFGreg |= SYS_CFG_RXM110K;
        reg16 >>= 3; // lde_replicaCoeff must be divided by 8
    }
    else
    {
        dw1000local.sysCFGreg &= (~SYS_CFG_RXM110K);
    }
    dw1000local.longFrames = config->phrMode;
    dw1000local.sysCFGreg &= ~SYS_CFG_PHR_MODE_11;
    dw1000local.sysCFGreg |= (SYS_CFG_PHR_MODE_11 &
                             (config->phrMode << SYS_CFG_PHR_MODE_SHFT));
    dwt_write32bitreg(SYS_CFG_ID, dw1000local.sysCFGreg);
    /* Set the lde_replicaCoeff */
    dwt_write16bitoffsetreg(LDE_IF_ID, LDE_REPC_OFFSET, reg16);
    _dwt_configlde(prfIndex);
    /* Configure PLL2/RF PLL block CFG/TUNE (for a given channel) */
    dwt_write32bitoffsetreg(FS_CTRL_ID, FS_PLLCFG_OFFSET,
                            fs_pll_cfg[chan_idx[chan]]);
    dwt_write8bitoffsetreg(FS_CTRL_ID, FS_PLLTUNE_OFFSET,
                           fs_pll_tune[chan_idx[chan]]);
    /* Configure RF RX blocks (for specified channel/bandwidth) */
    dwt_write8bitoffsetreg(RF_CONF_ID, RF_RXCTRLH_OFFSET, rx_config[bw]);
    /* Configure RF TX blocks (for specified channel and PRF)
     * Configure RF TX control */
    dwt_write32bitoffsetreg(RF_CONF_ID, RF_TXCTRL_OFFSET,
                            tx_config[chan_idx[chan]]);
    /* Configure the baseband parameters (for specified PRF, bit rate, PAC, and
     * SFD settings) */
```

```
// DTUNE0
dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_TUNE0b_OFFSET,
                        sftsh[config->dataRate][config->nsSFD]);
// DTUNE1
dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_TUNE1a_OFFSET, dtune1[prfIndex]);
if(config->dataRate == DWT_BR_110K)
{
    dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_TUNE1b_OFFSET, DRX_TUNE1b_110K);
}
else
{
    if(config->txPreambLength == DWT_PLEN_64)
    {
        dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_TUNE1b_OFFSET,
                                DRX_TUNE1b_6M8_PRE64);
        dwt_write8bitoffsetreg(DRX_CONF_ID, DRX_TUNE4H_OFFSET,
                               DRX_TUNE4H_PRE64);
    }
    else
    {
        dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_TUNE1b_OFFSET,
                                DRX_TUNE1b_850K_6M8);
        dwt_write8bitoffsetreg(DRX_CONF_ID, DRX_TUNE4H_OFFSET,
                               DRX_TUNE4H_PRE128PLUS);
    }
}
// DTUNE2
dwt_write32bitoffsetreg(DRX_CONF_ID, DRX_TUNE2_OFFSET,
                        digital_bb_config[prfIndex][config->rxPAC]);
// DTUNE3 (SFD timeout)
/* Don't allow 0 - SFD timeout will always be enabled */
if(config->sfdTO == 0)
{
    config->sfdTO = DWT_SFDTOC_DEF;
}
dwt_write16bitoffsetreg(DRX_CONF_ID, DRX_SFDTOC_OFFSET, config->sfdTO);
/* Configure AGC parameters */
dwt_write32bitoffsetreg( AGC_CFG_STS_ID, 0xC, agc_config.lo32);
dwt_write16bitoffsetreg( AGC_CFG_STS_ID, 0x4, agc_config.target[prfIndex]);
/* Set (non-standard) user SFD for improved performance */
if(config->nsSFD)
{
    /* Write non standard (DW) SFD length */
    dwt_write8bitoffsetreg(USR_SFD_ID, 0x00, dwnsSFDlen[config->dataRate]);
    nsSfd_result = 3;
    useDWnsSFD = 1;
}
regval = (CHAN_CTRL_TX_CHAN_MASK & (chan << CHAN_CTRL_TX_CHAN_SHIFT)) |
          // Transmit Channel
         (CHAN_CTRL_RX_CHAN_MASK & (chan << CHAN_CTRL_RX_CHAN_SHIFT)) |
          // Receive Channel
         (CHAN_CTRL_RXFPRF_MASK & (config->prf << CHAN_CTRL_RXFPRF_SHIFT)) |
          // RX PRF
```

```
            ((CHAN_CTRL_TNSSFD|CHAN_CTRL_RNSSFD) &
             (nsSfd_result<<CHAN_CTRL_TNSSFD_SHIFT)) |
             // nsSFD enable RX&TX
            (CHAN_CTRL_DWSFD & (useDWnsSFD<<CHAN_CTRL_DWSFD_SHIFT)) |
             // Use DW nsSFD
            (CHAN_CTRL_TX_PCOD_MASK & (config->txCode<<CHAN_CTRL_TX_PCOD_SHIFT)) |
             // TX Preamble Code
            (CHAN_CTRL_RX_PCOD_MASK & (config->rxCode<<CHAN_CTRL_RX_PCOD_SHIFT));
             // RX Preamble Code
    dwt_write32bitreg(CHAN_CTRL_ID,regval);
    /* Set up TX Preamble Size, PRF and Data Rate */
    dw1000local.txFCTRL=((config->txPreambLength|config->prf)<<TX_FCTRL_TXPRF_SHFT)
                        | (config->dataRate<<TX_FCTRL_TXBR_SHFT);
    dwt_write32bitreg(TX_FCTRL_ID, dw1000local.txFCTRL);
    /* The SFD transmit pattern is initialised by the DW1000 upon a user TX
     * request, but (due to an IC issue) it is not done for an auto-ACK TX. The
     * SYS_CTRL write below works around this issue, by simultaneously initiating
     * and aborting a transmission, which correctly initialises the SFD after its
     * configuration or reconfiguration. */
    dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET,
                           SYS_CTRL_TXSTRT | SYS_CTRL_TRXOFF);
    // Request TX start and TRX off at the same time
}

/*! ------------------------------------------------------------------------------
 * @fn dwt_setrxantennadelay()
 * @brief This API function writes the antenna delay (in time units) to the RX
 *        registers
 * @param rxDelay - this is the total (RX) antenna delay value, which will be
 *                  programmed into the RX register
 * @return none
 */
void dwt_setrxantennadelay(uint16 rxDelay)
{
    /* Set the RX antenna delay for auto TX timestamp adjustment */
    dwt_write16bitoffsetreg(LDE_IF_ID, LDE_RXANTD_OFFSET, rxDelay);
}

/*! ------------------------------------------------------------------------------
 * @fn dwt_settxantennadelay()
 * @brief This API function writes the antenna delay (in time units) to the TX
 *        registers
 * @param txDelay - this is the total (TX) antenna delay value, which will be
 *                  programmed into the TX delay register
 * @return none
 */
void dwt_settxantennadelay(uint16 txDelay)
{
    /* Set the TX antenna delay for auto TX timestamp adjustment */
    dwt_write16bitoffsetreg(TX_ANTD_ID, TX_ANTD_OFFSET, txDelay);
}

/*! ------------------------------------------------------------------------------
```

```
 * @fn dwt_writetxdata()
 * @brief This API function writes the supplied TX data into the DW1000's TX
 *        buffer. The input parameters are the data length in bytes and a pointer
 *        to those data bytes.
 * @param txFrameLength - This is the total frame length, including the 2 byte CRC.
 *                        NOTE: this is the length of TX message (including the 2
 *                              byte CRC) - max is 1023 standard PHR mode allows
 *                              up to 127 bytes if > 127 is programmed,
 *                              DWT_PHRMODE_EXT must be set in the phrMode
 *                              configuration; see dwt_configure function
 * @param txFrameBytes - Pointer to the user's buffer containing the data to send.
 * @param txBufferOffset - This specifies an offset in the DW1000's TX Buffer at
 *                         which to start writing data.
 * @return DWT_SUCCESS - success
 *         DWT_ERROR   - error
 */
int dwt_writetxdata(uint16 txFrameLength, uint8 *txFrameBytes,
                    uint16 txBufferOffset)
{
#ifdef DWT_API_ERROR_CHECK
    assert(txFrameLength >= 2);
    assert((dw1000local.longFrames && (txFrameLength <= 1023)) ||
           (txFrameLength <= 127));
    assert((txBufferOffset + txFrameLength) <= 1024);
#endif

    if ((txBufferOffset + txFrameLength) <= 1024)
    {
        /* Write the data to the IC TX buffer, (-2 bytes for auto generated CRC) */
        dwt_writetodevice(TX_BUFFER_ID, txBufferOffset, txFrameLength-2,
                          txFrameBytes);

        return DWT_SUCCESS;
    }
    else
    {
        return DWT_ERROR;
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_writetxfctrl()
 * @brief This API function configures the TX frame control register before the
 *        transmission of a frame
 * @param txFrameLength - this is the length of TX message (including the 2 byte
 *                        CRC) - max is 1023
 *                        NOTE: standard PHR mode allows up to 127 bytes if > 127
 *                              is programmed, DWT_PHRMODE_EXT must be set in the
 *                              phrMode configuration; see dwt_configure function
 * @param txBufferOffset - the offset in the tx buffer to start writing the data
 * @param ranging - 1 if this is a ranging frame, else 0
 * @return none
 */
```

```
void dwt_writetxfctrl(uint16 txFrameLength, uint16 txBufferOffset, int ranging)
{

#ifdef DWT_API_ERROR_CHECK
    assert((dw1000local.longFrames && (txFrameLength <= 1023)) ||
           (txFrameLength <= 127));
#endif

    /* Write the frame length to the TX frame control register */
    // dw1000local.txFCTRL has kept configured bit rate information
    uint32 reg32 = dw1000local.txFCTRL | txFrameLength |
                   (txBufferOffset<<TX_FCTRL_TXBOFFS_SHFT) |
                   (ranging<<TX_FCTRL_TR_SHFT);
    dwt_write32bitreg(TX_FCTRL_ID, reg32);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readrxdata()
 * @brief This is used to read the data from the RX buffer, from an offset
 *        location give by offset parameter
 * @param buffer - the buffer into which the data will be read
 * @param length - the length of data to read (in bytes)
 * @param rxBufferOffset - the offset in the rx buffer from which to read the data
 * @return none
 */
void dwt_readrxdata(uint8 *buffer, uint16 length, uint16 rxBufferOffset)
{
    dwt_readfromdevice(RX_BUFFER_ID,rxBufferOffset,length,buffer);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readtxtimestamp()
 * @brief This is used to read the TX timestamp (adjusted with the programmed
 *        antenna delay)
 * @param timestamp - a pointer to a 5-byte buffer which will store the read TX
 *                    timestamp time
 * @param the timestamp buffer will contain the value after the function call
 * @return none
 */
void dwt_readtxtimestamp(uint8 * timestamp)
{
    /* Read bytes directly into buffer */
    dwt_readfromdevice(TX_TIME_ID, TX_TIME_TX_STAMP_OFFSET, TX_TIME_TX_STAMP_LEN,
                       timestamp);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readtxtimestamplo32()
 * @brief This is used to read the low 32-bits of the TX timestamp (adjusted with
 *        the programmed antenna delay)
 * @param none
 * @return low 32-bits of TX timestamp
 */
```

```
uint32 dwt_readtxtimestamplo32(void)
{
    /* Read TX TIME as a 32-bit register to get the 4 lower bytes out of 5 */
    return dwt_read32bitreg(TX_TIME_ID);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readrxtimestamp()
 * @brief This is used to read the RX timestamp (adjusted time of arrival)
 * @param timestamp - a pointer to a 5-byte buffer which will store the read RX
 *                    timestamp time
 * @param the timestamp buffer will contain the value after the function call
 * @return none
 */
void dwt_readrxtimestamp(uint8 * timestamp)
{
    /* Get the adjusted time of arrival */
    dwt_readfromdevice(RX_TIME_ID, RX_TIME_RX_STAMP_OFFSET, RX_TIME_RX_STAMP_LEN,
                       timestamp);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readrxtimestamplo32()
 * @brief This is used to read the low 32-bits of the RX timestamp (adjusted with
 *        the programmed antenna delay)
 * @param none
 * @return low 32-bits of RX timestamp
 */
uint32 dwt_readrxtimestamplo32(void)
{
    /* Read RX TIME as a 32-bit register to get the 4 lower bytes out of 5 */
    return dwt_read32bitreg(RX_TIME_ID);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readsystime()
 * @brief This is used to read the system time
 * @param timestamp - a pointer to a 5-byte buffer which will store the read
 *                    system time
 * @param the timestamp buffer will contain the value after the function call
 * @return none
 */
void dwt_readsystime(uint8 * timestamp)
{
    dwt_readfromdevice(SYS_TIME_ID, SYS_TIME_OFFSET, SYS_TIME_LEN, timestamp);
}

/*! -------------------------------------------------------------------------------------
 * @fn dwt_writetodevice()
 * @brief This function is used to write to the DW1000 device registers
 * NOTES: 1. Firstly we create a header (the first byte is a header byte)
 *           a. check if sub index is used, if subindexing is used - set bit-6 to 1
 *           to signify that the sub-index address follows the register index byte
 *           b. set bit-7 (or with 0x80) for write operation
 *           c. if extended sub address index is used (i.e. if index > 127) set
 *           bit-7 of the first sub-index byte following the first header byte
 *        2. Write the header followed by the data bytes to the DW1000 device
 * @param recordNumber - ID of register file or buffer being accessed
 * @param index - byte index into register file or buffer being accessed
 * @param length - number of bytes being written
 * @param buffer - pointer to buffer containing the 'length' bytes to be written
 * @return nope
 */
void dwt_writetodevice
(
    uint16      recordNumber,
    uint16      index,
    uint32      length,
    const uint8 *buffer
)
{
    uint8 header[3]; // Buffer to compose header in
    int   cnt = 0; // Counter for length of header

#ifdef DWT_API_ERROR_CHECK
    assert(recordNumber <= 0x3F); // Record number is limited to 6-bits.
#endif

    /* Write message header selecting WRITE operation and addresses as
     * appropriate (this is one to three bytes long) */
    if (index == 0) // For index of 0, no sub-index is required
    {
        header[cnt++] = 0x80 | recordNumber; // Bit-7 is WRITE operation, bit-6
                                             zero=NO sub-addressing, bits 5-0 is
                                             reg file id
    }
    else
    {

#ifdef DWT_API_ERROR_CHECK
        assert((index <= 0x7FFF) && ((index + length) <= 0x7FFF));
        // Index and sub-addressable area are limited to 15-bits.
#endif

        header[cnt++] = 0xC0 | recordNumber; // Bit-7 is WRITE operation, bit-6
                                             one=sub-address follows, bits 5-0 is
                                             reg file id
        if (index <= 127) // For non-zero index < 127, just a single sub-index
                          byte is required
        {
            header[cnt++] = (uint8)index; // Bit-7 zero means no extension, bits
                                          6-0 is index.
        }
        else
        {
            header[cnt++] = 0x80 | (uint8)(index); // Bit-7 one means extended
                                                    index, bits 6-0 is low seven
                                                    bits of index.
            header[cnt++] =  (uint8) (index>>7); // 8-bit value = high eight bits
                                                  of index.
        }
    }
    /* Write it to the SPI */
    writetospi(cnt,header,length,buffer);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_readfromdevice()
 * @brief this function is used to read from the DW1000 device registers
 * NOTES: 1. Firstly we create a header (the first byte is a header byte)
 *           a. check if sub index is used, if subindexing is used - set bit-6 to 1
 *           to signify that the sub-index address follows the register index byte
 *           b. set bit-7 (or with 0x80) for write operation
 *           c. if extended sub address index is used (i.e. if index > 127) set
 *           bit-7 of the first sub-index byte following the first header byte
 *        2. Write the header followed by the data bytes to the DW1000 device
 *        3. Store the read data in the input buffer
 * @param recordNumber - ID of register file or buffer being accessed
 * @param index - byte index into register file or buffer being accessed
 * @param length - number of bytes being read
 * @param buffer - pointer to buffer in which to return the read data.
 * @return none
 */
void dwt_readfromdevice
(
    uint16 recordNumber,
    uint16 index,
    uint32 length,
    uint8  *buffer
)
{
    uint8 header[3]; // Buffer to compose header in
    int   cnt = 0; // Counter for length of header

#ifdef DWT_API_ERROR_CHECK
    assert(recordNumber <= 0x3F); // Record number is limited to 6-bits.
#endif

    /* Write message header selecting READ operation and addresses as appropriate
     * (this is one to three bytes long) */
    if (index == 0) // For index of 0, no sub-index is required
    {
        header[cnt++] = (uint8) recordNumber; // Bit-7 zero is READ operation,
                                               bit-6 zero=NO sub-addressing, bits
                                               5-0 is reg file id
    }
    else
    {
```

```
#ifdef DWT_API_ERROR_CHECK
        assert((index <= 0x7FFF) && ((index + length) <= 0x7FFF));
        // Index and sub-addressable area are limited to 15-bits.
#endif

        header[cnt++] = (uint8)(0x40 | recordNumber); // Bit-7 zero is READ
                                                       operation, bit-6 one=sub-
                                                       address follows, bits 5-0
                                                       is reg file id
        if (index <= 127) // For non-zero index < 127, just a single sub-index
                          byte is required
        {
            header[cnt++] = (uint8) index; // Bit-7 zero means no extension, bits
                                           6-0 form index.
        }
        else
        {
            header[cnt++] = 0x80 | (uint8)(index); // Bit-7 one means extended
                                                   index, bits 6-0 is low seven
                                                   bits of index.
            header[cnt++] = (uint8)(index >> 7); // 8-bit value = high eight bits
                                                 of index.
        }
    }
    /* Do the read from the SPI */
    readfromspi(cnt, header, length, buffer);  // result is stored in the buffer
}

/*! ---------------------------------------------------------------------------
 * @fn dwt_read32bitoffsetreg()
 * @brief This function is used to read 32-bit value from the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @return regval - a 32 bit register value
 */
uint32 dwt_read32bitoffsetreg(int regFileID,int regOffset)
{
    uint32 regval = 0;
    int    j;
    uint8  buffer[4];
    dwt_readfromdevice(regFileID,regOffset,4,buffer); // Read 4 bytes (32-bits)
                                                       register into buffer
    for (j = 3; j >= 0; j--)
    {
        regval = (regval << 8) + buffer[j];
    }

    return regval;
}

/*! ---------------------------------------------------------------------------
```

```
 * @fn dwt_read16bitoffsetreg()
 * @brief This function is used to read 16-bit value from the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @return regval - 16 bit register value
 */
uint16 dwt_read16bitoffsetreg(int regFileID,int regOffset)
{
    uint16 regval = 0;
    uint8  buffer[2];
    dwt_readfromdevice(regFileID,regOffset,2,buffer); // Read 2 bytes (16-bits)
                                                       register into buffer
    regval = (buffer[1]<<8) + buffer[0];

    return regval;
}

/*! ------------------------------------------------------------------------------
 * @fn dwt_read8bitoffsetreg()
 * @brief This function is used to read an 8-bit value from the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @return regval - 8-bit register value
 */
uint8 dwt_read8bitoffsetreg(int regFileID, int regOffset)
{
    uint8 regval;
    dwt_readfromdevice(regFileID, regOffset, 1, &regval);

    return regval;
}

/*! ------------------------------------------------------------------------------
 * @fn dwt_write8bitoffsetreg()
 * @brief This function is used to write an 8-bit value to the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @param regval - the value to write
 * @return none
 */
void dwt_write8bitoffsetreg(int regFileID, int regOffset, uint8 regval)
{
    dwt_writetodevice(regFileID, regOffset, 1, &regval);
}

/*! ------------------------------------------------------------------------------
 * @fn dwt_write16bitoffsetreg()
 * @brief This function is used to write 16-bit value to the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @param regval - the value to write
 * @return none
 */
void dwt_write16bitoffsetreg(int regFileID,int regOffset,uint16 regval)
{
    uint8   buffer[2];
    buffer[0] = regval & 0xFF;
    buffer[1] = regval>>8;
    dwt_writetodevice(regFileID,regOffset,2,buffer);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_write32bitoffsetreg()
 * @brief This function is used to write 32-bit value to the DW1000 device
 *        registers
 * @param regFileID - ID of register file or buffer being accessed
 * @param regOffset - the index into register file or buffer being accessed
 * @param regval - the value to write
 * @return none
 */
void dwt_write32bitoffsetreg(int regFileID,int regOffset,uint32 regval)
{
    int   j;
    uint8 buffer[4];
    for (j = 0; j < 4; j++)
    {
        buffer[j] = regval & 0xff;
        regval >>= 8;
    }
    dwt_writetodevice(regFileID,regOffset,4,buffer);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_enableframefilter()
 * @brief This is used to enable the frame filtering - (the default option is to
 *        accept any data and ACK frames with correct destination address
 * @param bitmask - enables/disables the frame filtering options according to
 *                  DWT_FF_NOTYPE_EN        0x000   no frame types allowed
 *                  DWT_FF_COORD_EN         0x002   behave as coordinator (can
 *                                                  receive frames with no
 *                                                  destination address (PAN ID
 *                                                  has to match))
 *                  DWT_FF_BEACON_EN        0x004   beacon frames allowed
 *                  DWT_FF_DATA_EN          0x008   data frames allowed
 *                  DWT_FF_ACK_EN           0x010   ack frames allowed
 *                  DWT_FF_MAC_EN           0x020   mac control frames allowed
 *                  DWT_FF_RSVD_EN          0x040   reserved frame types allowed
 * @return none
 */
void dwt_enableframefilter(uint16 enable)
{
    // Read sysconfig register
    uint32 sysconfig = SYS_CFG_MASK & dwt_read32bitreg(SYS_CFG_ID);
    if(enable)
    {
        /* Enable frame filtering and configure frame types */
        sysconfig &= ~(SYS_CFG_FF_ALL_EN); // Clear all
        sysconfig |= (enable & SYS_CFG_FF_ALL_EN) | SYS_CFG_FFE;
    }
    else
    {
        sysconfig &= ~(SYS_CFG_FFE);
    }
    dw1000local.sysCFGreg = sysconfig;
    dwt_write32bitreg(SYS_CFG_ID,sysconfig);
}

/*! ------------------------------------------------------------------------------------------------------------------
 * @fn dwt_setpanid()
 * @brief This is used to set the PAN ID
 * @param panID - this is the PAN ID
 * @return none
 */
void dwt_setpanid(uint16 panID)
{
    /* PAN ID is high 16 bits of register */
    dwt_write16bitoffsetreg(PANADR_ID, PANADR_PAN_ID_OFFSET, panID);
}

/*! ------------------------------------------------------------------------------------------------------------------
 * @fn dwt_setaddress16()
 * @brief This is used to set 16-bit (short) address
 * @param shortAddress - this sets the 16 bit short address
 * @return none
 */
void dwt_setaddress16(uint16 shortAddress)
{
    /* Short address into low 16 bits */
    dwt_write16bitoffsetreg(PANADR_ID, PANADR_SHORT_ADDR_OFFSET, shortAddress);
}

/*! ------------------------------------------------------------------------------------------------------------------
 * @fn dwt_seteui()
 * @brief This is used to set the EUI 64-bit (long) address
 * @param eui64 - this is the pointer to a buffer that contains the 64bit address
 * @return none
 */
void dwt_seteui(uint8 *eui64)
{
    dwt_writetodevice(EUI_64_ID, EUI_64_OFFSET, EUI_64_LEN, eui64);
}

/*! ------------------------------------------------------------------------------------------------------------------
 * @fn dwt_geteui()
 * @brief This is used to get the EUI 64-bit from the DW1000
 * @param eui64 - this is the pointer to a buffer that will contain the read 64-
 *                bit EUI value
 * @return none
 */
void dwt_geteui(uint8 *eui64)
{
    dwt_readfromdevice(EUI_64_ID, EUI_64_OFFSET, EUI_64_LEN, eui64);
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_otpread()
 * @brief function to read the OTP memory. Ensure that MR,MRa,MRb are reset to 0.
 * @param address - address to read at
 * @return ret_data - the 32bit of read data
 */
uint32 _dwt_otpread(uint32 address)
{
    uint32 ret_data;
    // Write the address
    dwt_write16bitoffsetreg(OTP_IF_ID, OTP_ADDR, address);
    // Perform OTP Read - Manual read mode has to be set
    dwt_write8bitoffsetreg(OTP_IF_ID, OTP_CTRL, OTP_CTRL_OTPREAD |
                           OTP_CTRL_OTPRDEN);
    dwt_write8bitoffsetreg(OTP_IF_ID, OTP_CTRL, 0x00); // OTPREAD is self clearing
                                                          but OTPRDEN is not
    // Read read data, available 40ns after rising edge of OTP_READ
    ret_data = dwt_read32bitoffsetreg(OTP_IF_ID, OTP_RDAT);

    // Return the 32bit of read data
    return ret_data;
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_otpprogword32()
 * @brief function to program the OTP memory. Ensure MR,MRa,MRb are reset to 0.
 *        VNM Charge pump needs to be enabled (see _dwt_otpsetmrregs)
 *        Note the address is only 11 bits long.
 * @param address - address to read at
 * @return DWT_SUCCESS - success
 *         DWT_ERROR   - error
 */
uint32 _dwt_otpprogword32(uint32 data, uint16 address)
{
    uint8 rd_buf[1];
    uint8 wr_buf[4];
    uint8 otp_done;
    /* Read status register */
    dwt_readfromdevice(OTP_IF_ID, OTP_STAT, 1, rd_buf);
    if((rd_buf[0] & 0x02) != 0x02)
    {
        return DWT_ERROR;
    }
    /* Write the data */
    wr_buf[3] = (data>>24) & 0xff;
    wr_buf[2] = (data>>16) & 0xff;
    wr_buf[1] = (data>>8)  & 0xff;
    wr_buf[0] = data & 0xff;
    dwt_writetodevice(OTP_IF_ID, OTP_WDAT, 4, wr_buf);
    /* Write the address [10:0] */
    wr_buf[1] = (address>>8) & 0x07;
    wr_buf[0] = address & 0xff;
    dwt_writetodevice(OTP_IF_ID, OTP_ADDR, 2, wr_buf);
    /* Enable Sequenced programming */
    wr_buf[0] = OTP_CTRL_OTPPROG;
    dwt_writetodevice(OTP_IF_ID, OTP_CTRL, 1, wr_buf);
    wr_buf[0] = 0x00; // And clear
    dwt_writetodevice(OTP_IF_ID, OTP_CTRL, 1, wr_buf);
    /* WAIT for status to flag PRGM OK. */
    otp_done = 0;
    while(otp_done == 0)
    {
        deca_sleep(1);
        dwt_readfromdevice(OTP_IF_ID, OTP_STAT, 1, rd_buf);

        if((rd_buf[0] & 0x01) == 0x01)
        {
            otp_done = 1;
        }
    }

    return DWT_SUCCESS;
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_aonconfigupload()
 * @brief This function uploads always on (AON) configuration, as set in the
 *        AON_CFG0_OFFSET register.
 * @param none
 * @return none
 */
void _dwt_aonconfigupload(void)
{
    dwt_write8bitoffsetreg(AON_ID, AON_CTRL_OFFSET, AON_CTRL_UPL_CFG);
    dwt_write8bitoffsetreg(AON_ID, AON_CTRL_OFFSET, 0x00); // Clear the register
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_aonarrayupload()
 * @brief This function uploads always on (AON) data array and configuration.
 *        Thus if this function is used, then _dwt_aonconfigupload is not
 *        necessary. The DW1000 will go so SLEEP straight after this if the
 *        DWT_SLP_EN has been set.
 * @param none
 * @return none
 */
void _dwt_aonarrayupload(void)
```

```
{
    dwt_write8bitoffsetreg(AON_ID, AON_CTRL_OFFSET, 0x00); // Clear the register
    dwt_write8bitoffsetreg(AON_ID, AON_CTRL_OFFSET, AON_CTRL_SAVE);
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_configlde()
 * @brief Configure LDE algorithm parameters
 * @param prf - this is the PRF index (0 or 1) 0 corresponds to 16 and 1 to 64 PRF
 * @return none
 */
void _dwt_configlde(int prfIndex)
{
    dwt_write8bitoffsetreg(LDE_IF_ID, LDE_CFG1_OFFSET, LDE_PARAM1);
    // 8-bit configuration register
    if(prfIndex)
    {
        dwt_write16bitoffsetreg(LDE_IF_ID, LDE_CFG2_OFFSET, (uint16)
                                LDE_PARAM3_64);
        // 16-bit LDE configuration tuning register
    }
    else
    {
        dwt_write16bitoffsetreg(LDE_IF_ID, LDE_CFG2_OFFSET, (uint16)
                                LDE_PARAM3_16);
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_loaducodefromrom()
 * @brief Load ucode from OTP MEMORY or ROM
 * @param none
 * @return none
 */
void _dwt_loaducodefromrom(void)
{
    /* Set up clocks */
    _dwt_enableclocks(FORCE_LDE);
    /* Kick off the LDE load */
    dwt_write16bitoffsetreg(OTP_IF_ID, OTP_CTRL, OTP_CTRL_LDELOAD);
    // Set load LDE kick bit
    deca_sleep(1); // Allow time for code to upload (should take up to 120 us)
    /* Default clocks (ENABLE_ALL_SEQ) */
    _dwt_enableclocks(ENABLE_ALL_SEQ); // Enable clocks for sequencing
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setrxaftertxdelay()
 * @brief This sets the receiver turn on delay time after a transmission of a frame
 * @param rxDelayTime - (20 bits) the delay is in UWB microseconds
 * @return none
 */
void dwt_setrxaftertxdelay(uint32 rxDelayTime)
```

```
{
    uint32 val = dwt_read32bitreg(ACK_RESP_T_ID); // Read ACK_RESP_T_ID register
    val &= ~(ACK_RESP_T_W4R_TIM_MASK); // Clear the timer (19:0)
    val |= (rxDelayTime & ACK_RESP_T_W4R_TIM_MASK); // In UWB microseconds
                                                     (e.g. turn the receiver on
                                                     20uus after TX)
    dwt_write32bitreg(ACK_RESP_T_ID, val);
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_enableclocks()
 * @brief function to enable/disable clocks to particular digital blocks/system
 * @param clocks - set of clocks to enable/disable
 * @return none
 */
void _dwt_enableclocks(int clocks)
{
    uint8 reg[2];
    dwt_readfromdevice(PMSC_ID, PMSC_CTRL0_OFFSET, 2, reg);
    switch(clocks)
    {
        case ENABLE_ALL_SEQ:
        {
            reg[0] = 0x00;
            reg[1] = reg[1] & 0xfe;
        }
        break;
        case FORCE_SYS_XTI:
        {
            // System and RX
            reg[0] = 0x01 | (reg[0] & 0xfc);
        }
        break;
        case FORCE_SYS_PLL:
        {
            // System
            reg[0] = 0x02 | (reg[0] & 0xfc);
        }
        break;
        case READ_ACC_ON:
        {
            reg[0] = 0x48 | (reg[0] & 0xb3);
            reg[1] = 0x80 | reg[1];
        }
        break;
        case READ_ACC_OFF:
        {
            reg[0] = reg[0] & 0xb3;
            reg[1] = 0x7f & reg[1];
        }
        break;
        case FORCE_OTP_ON:
        {
            reg[1] = 0x02 | reg[1];
        }
        break;
        case FORCE_OTP_OFF:
        {
            reg[1] = reg[1] & 0xfd;
        }
        break;
        case FORCE_TX_PLL:
        {
            reg[0] = 0x20 | (reg[0] & 0xcf);
        }
        break;
        case FORCE_LDE:
        {
            reg[0] = 0x01;
            reg[1] = 0x03;
        }
        break;
        default:
        break;
    }
    /* Need to write lower byte separately before setting the higher byte(s) */
    dwt_writetodevice(PMSC_ID, PMSC_CTRL0_OFFSET, 1, &reg[0]);
    dwt_writetodevice(PMSC_ID, 0x1, 1, &reg[1]);
}

/*! ------------------------------------------------------------------------------------
 * @fn _dwt_disablesequencing()
 * @brief This function disables the TX blocks sequencing, it disables PMSC
 *        control of RF blocks, system clock is also set to XTAL
 * @param none
 * @return none
 */
void _dwt_disablesequencing(void) // Disable sequencing and go to state "INIT"
{
    _dwt_enableclocks(FORCE_SYS_XTI); // Set system clock to XTI
    // Disable PMSC ctrl of RF and RX clk blocks
    dwt_write16bitoffsetreg(PMSC_ID, PMSC_CTRL1_OFFSET,
                            PMSC_CTRL1_PKTSEQ_DISABLE);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setdelayedtrxtime()
 * @brief This API function configures the delayed transmit time or the delayed
 *        RX on time
 * @param starttime - the TX/RX start time (the 32 bits should be the high 32
 *                    bits of the system time at which to send the message, or at
 *                    which to turn on the receiver)
 * @return none
 */
void dwt_setdelayedtrxtime(uint32 starttime)
{
    dwt_write32bitoffsetreg(DX_TIME_ID, 1, starttime);
    // Write at offset 1 as the lower 9 bits of this register are ignored
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_starttx()
 * @brief This call initiates the transmission, input parameter indicates which
 *        TX mode is used see below
 * @param mode - if 0 immediate TX (no response expected)
 *               if 1 delayed TX (no response expected)
 *               if 2 immediate TX (response expected - so the receiver will be
 *                    automatically turned on after TX is done)
 *               if 3 delayed TX (response expected - so the receiver will be
 *                    automatically turned on after TX is done)
 * @returns DWT_SUCCESS - success
 *          DWT_ERROR   - error (e.g. a delayed transmission will fail if the
 *                               delayed time has passed)
 */
int dwt_starttx(uint8 mode)
{
    int retval = DWT_SUCCESS;
    uint8 temp = 0x00;
    uint16 checkTxOK = 0;
    if (mode & DWT_RESPONSE_EXPECTED)
    {
        temp = (uint8)SYS_CTRL_WAIT4RESP ; // Set wait4response bit
        dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET, temp);
        dw1000local.wait4resp = 1;
        //print_string("RESPONSE EXPECTED");
    }
    if (mode & DWT_START_TX_DELAYED)
    {
        /* Both SYS_CTRL_TXSTRT and SYS_CTRL_TXDLYS to correctly enable TX */
        temp |= (uint8)(SYS_CTRL_TXDLYS | SYS_CTRL_TXSTRT) ;
        dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET, temp);
        checkTxOK = dwt_read16bitoffsetreg(SYS_STATUS_ID, 3);
        // Read at offset 3 to get the upper 2 bytes out of 5

        if ((checkTxOK & SYS_STATUS_TXERR) == 0)
        // Transmit Delayed Send set over Half a Period away or Power Up error
           (there is enough time to send but not to power up individual blocks).
        {
            retval = DWT_SUCCESS; // All okay
        }
        else
        {
            /* Take DSHP set to Indicate that the TXDLYS was set too late for the
             * specified DX_TIME. */
            // Remedial Action - (a) cancel delayed send
            temp = (uint8)SYS_CTRL_TRXOFF; // This assumes the bit is in the
                                              lowest byte
            dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET, temp);
            /* Note event Delayed TX Time too Late */
            // Could fall through to start a normal send (below) just sending
            late instead return and assume return value of 1 will be used to
            detect and recover from the issue.
            dw1000local.wait4resp = 0;
            retval = DWT_ERROR ; // Failed!
        }
    }
    else
    {
        temp |= (uint8)SYS_CTRL_TXSTRT;
        dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET, temp);
    }
    /*if (retval==0){print_string("SUCCESS");}
    else {print_string("FAILURE");} */

    return retval;
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_syncrxbufptrs()
 * @brief this function synchronizes rx buffer pointers
 *        need to make sure that the host/IC buffer pointers are aligned before
 *        starting RX
 * @param none
 * @return none
 */
void dwt_syncrxbufptrs(void)
{
    uint8 buff;
    /* Make sure that the host/IC buffer pointers are aligned before starting RX */
    buff = dwt_read8bitoffsetreg(SYS_STATUS_ID, 3); // Read 1 byte at offset 3 to
                                                        get the 4th byte out of 5
    if((buff & (SYS_STATUS_ICRBP >> 24)) !=   // IC side Receive Buffer Pointer
       ((buff & (SYS_STATUS_HSRBP>>24))<<1) ) // Host Side Receive Buffer Pointer
    {
        dwt_write8bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_HRBT_OFFSET , 0x01);
        // Need to swap RX buffer status reg (write one to toggle internally)
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_rxenable()
 * @brief This call turns on the receiver, can be immediate or delayed (depending
 *        on the mode parameter). In the case of a "late" error the receiver will
 *        only be turned on if the DWT_IDLE_ON_DLY_ERR is not set.
 *        The receiver will stay turned on, listening to any messages until it
 *        either receives a good frame, an error (CRC, PHY header, Reed Solomon)
 *        or it times out (SFD, Preamble or Frame).
 * @param mode - this can be one of the following allowed values:
 *               DWT_START_RX_IMMEDIATE 0 enables receiver immediately
 *               DWT_START_RX_DELAYED   1 sets up delayed RX, if "late" error
 *               triggers, then the RX will be enabled immediately
 *               (DWT_START_RX_DELAYED | DWT_IDLE_ON_DLY_ERR) 3 used to disable
 *                re-enabling of receiver if delayed RX failed due to "late" error
 *                (DWT_START_RX_IMMEDIATE | DWT_NO_SYNC_PTRS)  4 used to re-enable
 *                RX without trying to sync IC and host side buffer pointers,
 *                typically when performing manual RX re-enabling in double
 *                buffering mode
 * @return DWT_SUCCESS - success
 *         DWT_ERROR   - error (e.g. a delayed receive enable will be too far in
 *                               the future if delayed time has passed)
 */
int dwt_rxenable(int mode)
{
    uint16 temp;
    uint8 temp1;
    if ((mode & DWT_NO_SYNC_PTRS) == 0)
    {
        dwt_syncrxbufptrs();
    }
    temp = (uint16)SYS_CTRL_RXENAB;
    if (mode & DWT_START_RX_DELAYED)
    {
        temp |= (uint16)SYS_CTRL_RXDLYE;
    }
    dwt_write16bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET, temp);
    if (mode & DWT_START_RX_DELAYED) // check for errors
    {
        temp1 = dwt_read8bitoffsetreg(SYS_STATUS_ID, 3);
        // Read 1 byte at offset 3 to get the 4th byte out of 5
        if ((temp1 & (SYS_STATUS_HPDWARN >> 24)) != 0)
        // if delay has passed do immediate RX unless DWT_IDLE_ON_DLY_ERR is true
        {
            if((mode & DWT_IDLE_ON_DLY_ERR) == 0)
            // if DWT_IDLE_ON_DLY_ERR not set then re-enable receiver
            {
                dwt_write16bitoffsetreg(SYS_CTRL_ID, SYS_CTRL_OFFSET,
                                        SYS_CTRL_RXENAB);
            }
            return DWT_ERROR; // return warning indication
        }
    }
    return DWT_SUCCESS;
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_setrxtimeout()
 * @brief This call enables RX timeout (SY_STAT_RFTO event)
 * @param time - how long the receiver remains on from the RX enable command
 *               The time parameter used here is in 1.0256 us (512/499.2MHz) units
 *               If set to 0 the timeout is disabled.
 * @return none
 */
void dwt_setrxtimeout(uint16 time)
{
    uint8 temp;
    temp = dwt_read8bitoffsetreg(SYS_CFG_ID, 3); // Read at offset 3 to get the
                                                    upper byte only
    if(time > 0)
    {
        dwt_write16bitoffsetreg(RX_FWTO_ID, RX_FWTO_OFFSET, time);
        temp |= (uint8)(SYS_CFG_RXWTOE>>24); // Shift RXWTOE mask as we read the
                                             upper byte only
        // OR in 32bit value (1 bit set), I know this is in high byte.
        dw1000local.sysCFGreg |= SYS_CFG_RXWTOE;
        dwt_write8bitoffsetreg(SYS_CFG_ID, 3, temp); // Write at offset 3 to
                                                       write the upper byte only
    }
    else
    {
        temp &=~((uint8)(SYS_CFG_RXWTOE>>24)); // Shift RXWTOE mask as we read
                                                 the upper byte only
        // AND in inverted 32bit value (1 bit clear), note this is in high byte.
        dw1000local.sysCFGreg &=~(SYS_CFG_RXWTOE);
        dwt_write8bitoffsetreg(SYS_CFG_ID, 3, temp); // Write at offset 3 to
                                                       write the upper byte only
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_rxreset()
 * @brief This function resets the receiver of the DW1000
 * @param none
 * @return none
 */
void dwt_rxreset(void)
{
    /* Set RX reset */
    dwt_write8bitoffsetreg(PMSC_ID, PMSC_CTRL0_SOFTRESET_OFFSET,
                           PMSC_CTRL0_RESET_RX);
    /* Clear RX reset */
    dwt_write8bitoffsetreg(PMSC_ID, PMSC_CTRL0_SOFTRESET_OFFSET,
                           PMSC_CTRL0_RESET_CLEAR);
}

/*! ------------------------------------------------------------------------------------
 * @fn dwt_softreset()
 * @brief This function resets the DW1000
 * @param none
 * @return none
 */
void dwt_softreset(void)
{
    _dwt_disablesequencing();
    /* Clear any AON auto download bits (as reset will trigger AON download) */
    dwt_write16bitoffsetreg(AON_ID, AON_WCFG_OFFSET, 0x00);
    /* Clear the wake-up configuration */
    dwt_write8bitoffsetreg(AON_ID, AON_CFG0_OFFSET, 0x00);
    /* Upload the new configuration */
    _dwt_aonarrayupload();
    /* Reset HIF, TX, RX and PMSC */
    dwt_write8bitoffsetreg(PMSC_ID, PMSC_CTRL0_SOFTRESET_OFFSET,
                           PMSC_CTRL0_RESET_ALL);
    /* DW1000 needs a 10us sleep to let clk PLL lock after reset - the PLL will
     * automatically lock after the reset. Could also have polled the PLL lock
     * flag, but then the SPI needs to be < 3MHz!! So a simple delay is easier */
    deca_sleep(1);
    /* Clear reset */
    dwt_write8bitoffsetreg(PMSC_ID, PMSC_CTRL0_SOFTRESET_OFFSET,
                           PMSC_CTRL0_RESET_CLEAR);
    dw1000local.wait4resp = 0;
}

/*! ---------------------------------------------------------------------------
 * @fn dwt_setxtaltrim()
 * @brief This is used to adjust the crystal frequency
 * @param value - crystal trim value (in range 0x0 to 0x1F), 31 steps (~1.5ppm
 *                per step)
 * @return none
 */
void dwt_setxtaltrim(uint8 value)
{
    /* The 3 MSb in this 8-bit register must be kept to 0b011 to avoid any
     * malfunction. */
    uint8 reg_val = (3 << 5) | (value & FS_XTALT_MASK);
    dwt_write8bitoffsetreg(FS_CTRL_ID, FS_XTALT_OFFSET, reg_val);
}
```

deca_params_init.c

```
/*! --------------------------------------------------------------------------
 * @file deca_params_init.c
 * @brief DW1000 configuration parameters
 * Copyright 2013 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#include <stdio.h>
#include <stdlib.h>

#include "deca_regs.h"
#include "deca_device_api.h"
#include "deca_param_types.h"


//-----------------------------------------
/* map the channel number to the index in the configuration arrays below
 * 0th element is chan 1, 1st is chan 2, 2nd is chan 3, 3rd is chan 4, 4th is
 * chan 5, 5th is chan 7 */
const uint8 chan_idx[NUM_CH_SUPPORTED] = {0, 0, 1, 2, 3, 4, 0, 5};
//-----------------------------------------
```

```
const uint32 tx_config[NUM_CH] =
{
    RF_TXCTRL_CH1,
    RF_TXCTRL_CH2,
    RF_TXCTRL_CH3,
    RF_TXCTRL_CH4,
    RF_TXCTRL_CH5,
    RF_TXCTRL_CH7,
};

/* Frequency Synthesiser - PLL configuration */
const uint32 fs_pll_cfg[NUM_CH] =
{
    FS_PLLCFG_CH1,
    FS_PLLCFG_CH2,
    FS_PLLCFG_CH3,
    FS_PLLCFG_CH4,
    FS_PLLCFG_CH5,
    FS_PLLCFG_CH7
};

/* Frequency Synthesiser - PLL tuning */
const uint8 fs_pll_tune[NUM_CH] =
{
    FS_PLLTUNE_CH1,
    FS_PLLTUNE_CH2,
    FS_PLLTUNE_CH3,
    FS_PLLTUNE_CH4,
    FS_PLLTUNE_CH5,
    FS_PLLTUNE_CH7
};

/* Bandwidth configuration */
const uint8 rx_config[NUM_BW] =
{
    RF_RXCTRLH_NBW,
    RF_RXCTRLH_WBW
};

const agc_cfg_struct agc_config =
{
    AGC_TUNE2_VAL,
    {AGC_TUNE1_16M, AGC_TUNE1_64M}  //adc target
};

/* DW non-standard SFD length for 110k, 850k and 6.81M */
const uint8 dwnsSFDlen[NUM_BR] =
{
    DW_NS_SFD_LEN_110K,
    DW_NS_SFD_LEN_850K,
    DW_NS_SFD_LEN_6M8
};
```

```
/* SFD Threshold */
const uint16 sftsh[NUM_BR][NUM_SFD] =
{
    {
        DRX_TUNE0b_110K_STD,
        DRX_TUNE0b_110K_NSTD
    },
    {
        DRX_TUNE0b_850K_STD,
        DRX_TUNE0b_850K_NSTD
    },
    {
        DRX_TUNE0b_6M8_STD,
        DRX_TUNE0b_6M8_NSTD
    }
};

const uint16 dtune1[NUM_PRF] =
{
    DRX_TUNE1a_PRF16,
    DRX_TUNE1a_PRF64
};

const uint32 digital_bb_config[NUM_PRF][NUM_PACS] =
{
    {
        DRX_TUNE2_PRF16_PAC8,
        DRX_TUNE2_PRF16_PAC16,
        DRX_TUNE2_PRF16_PAC32,
        DRX_TUNE2_PRF16_PAC64
    },
    {
        DRX_TUNE2_PRF64_PAC8,
        DRX_TUNE2_PRF64_PAC16,
        DRX_TUNE2_PRF64_PAC32,
        DRX_TUNE2_PRF64_PAC64
    }
};

const uint16 lde_replicaCoeff[PCODES] =
{
    0, // No preamble code 0
    LDE_REPC_PCODE_1,
    LDE_REPC_PCODE_2,
    LDE_REPC_PCODE_3,
    LDE_REPC_PCODE_4,
    LDE_REPC_PCODE_5,
    LDE_REPC_PCODE_6,
    LDE_REPC_PCODE_7,
    LDE_REPC_PCODE_8,
    LDE_REPC_PCODE_9,
    LDE_REPC_PCODE_10,
    LDE_REPC_PCODE_11,
    LDE_REPC_PCODE_12,
    LDE_REPC_PCODE_13,
    LDE_REPC_PCODE_14,
    LDE_REPC_PCODE_15,
    LDE_REPC_PCODE_16,
    LDE_REPC_PCODE_17,
    LDE_REPC_PCODE_18,
    LDE_REPC_PCODE_19,
    LDE_REPC_PCODE_20,
    LDE_REPC_PCODE_21,
    LDE_REPC_PCODE_22,
    LDE_REPC_PCODE_23,
    LDE_REPC_PCODE_24
};
```

---

deca_sleep.c

```
/*! ----------------------------------------------------------------------------
 * @file deca_sleep.c
 * @brief platform dependent sleep implementation
 * Copyright 2015 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 *  edited 12.2.17
 */

#include "sleep.h"
#include "SDlib16.h"

/*! ------------------------------------------------------------------------------
 * @fn deca_sleep()
 * @brief Wait for a given amount of time.
 *        NOTES: The body of this function is platform specific
 * @param time_ms - time to wait in milliseconds
 * @return none
 */
void deca_sleep(unsigned int time_ms)
{
    sleep_ms(time_ms);
}

/*! ------------------------------------------------------------------------------
 * @fn sleep_ms()
 * @brief Wait for a given amount of time.
 *        NOTES: The body of this function is platform specific
 * @param time_ms - time to wait in milliseconds
 * @return none
 */
void sleep_ms(unsigned int time_ms)
{
    delay_ms((unsigned long) time_ms); //Function from SDlib16.
}
```

deca_spi.c

```c
/*! ------------------------------------------------------------------------------
 * @file deca_spi.c
 * @brief SPI access functions
 * Copyright 2013 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 3.3.17
 */

#include <string.h>

#include "deca_spi.h"
#include "deca_device_api.h"
#include "port.h"
#include <xc.h>

/*! -------------------------------------------------------------------------------
 * @fn openspi()
 * @brief Open and initialize access to the SPI device
 * @param none
 * @return 0  - success
 *         -1 - error
 */
int openspi(/*SPI_TypeDef* SPIx*/)
{
    SPI_ON = 1;
    return 0;
}

/*! -------------------------------------------------------------------------------
 * @fn closespi()
 * @brief close the SPI device
 * @param none
 * @return 0  - success
 *         -1 - error
 */
int closespi(void)
{
    while (SPI_BUSY); //wait for tx buffer to empty
    SPI_ON = 0;
    return 0;
}

/*! -------------------------------------------------------------------------------
 * @fn writetospi()
 * @brief Low level abstract function to write to the SPI. Takes two separate
 *        byte buffers for write header and write data, returns 0 if read and -1
 *        if there was an error
 *        NOTES: The body of this function is platform specific
 * @param headerLength - number of bytes header being written
 * @param headerBuffer - pointer to buffer containing the 'headerLength' bytes of
 *                        header to be written
 * @param bodylength - number of bytes data being written
 * @param bodyBuffer - pointer to buffer containing the 'bodylength' bytes od
 *                      data to be written
 * @return DWT_SUCCESS - success
 *         DWT_ERROR   - error
 */
#pragma GCC optimize ("O3")
int writetospi(uint16 headerLength, const uint8 *headerBuffer, uint32 bodylength,
               const uint8 *bodyBuffer)
{
    int i = 0;
    SSLAT = 0;
    for(i = 0; i < headerLength; i++)
    {
        spi_transaction(headerBuffer[i]);
    }
    for(i = 0; i < bodylength; i++)
    {
        spi_transaction(bodyBuffer[i]);
    }
    SSLAT = 1;
    return 0;
}

/*! ------------------------------------------------------------------------------------
 * @fn readfromspi()
 * @brief Low level abstract function to read from the SPI. Takes two separate
 *        byte buffers for write header and read data, returns the offset into
 *        read buffer where first byte of read data may be found, or -1 if there
 *        was an error
 *        NOTES: The body of this function is defined in deca_spi.c and is
 *               platform specific
 * @param headerLength - number of bytes header to write
 * @param headerBuffer - pointer to buffer containing the 'headerLength' bytes of
 *                        header to write
 * @param readlength - number of bytes data being read
 * @param readBuffer - pointer to buffer containing to return the data (NB: size
 *                      required = headerLength + readlength)
 * @return DWT_SUCCESS - success (and gives the position in the buffer at which
 *                        data begins)
 *         DWT_ERROR   - error
 */
int readfromspi(uint16 headerLength, const uint8 *headerBuffer, uint32
                readlength, uint8 *readBuffer)
{
    int i = 0;
    /* Wait for SPIx Tx buffer empty */
    SSLAT = 0;
    for(i = 0; i < headerLength; i++)
    {
         readBuffer[0] = spi_transaction(headerBuffer[i]);
    }
    for(i = 0; i < readlength; i++)
    {
        readBuffer[i] = spi_transaction(0);
    }
    SSLAT = 1;
    return 0;
}

/*! ------------------------------------------------------------------------------
 * @fn spi_transaction()
 * @brief One byte communication with SPI
 * @param data - string to be sent
 * @return string received
 */
unsigned char spi_transaction(unsigned char data)
{
    IFS1bits.SPI4TXIF = 0;
    SPI4BUF = data;
    while(!SPI4STATbits.SPIRBF);
    return SPI4BUF;
}
```

functions17.c

```
/*
 * @file functions17.c
 * @author Katherine Sanders
 * @brief Configuration for RFPS board
 *
 * edited 25.4.17
 */

#include "configbits-16ex8.h"
#include "deca_device_api.h"
#include "deca_regs.h"
#include "functions17.h"
#include "port.h"
#include "SDlib16.h"
#include "sleep.h"

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//#define DEBUG

/*!------------------------------------------------------------------------------
 * @fn getu_3()
 * @breif Get character from UART3
 * @param value received from/ displayed via UART3
 * @return character gotten from UART3
 */
unsigned char getu_3(void)
{
    while(!U3STAbits.URXDA);    //data received yet
    return(U3RXREG);
}

/*!--------------------------------------------------------------------------------
 * @fn putu_3()
 * @breif Send character to UART3
 * @param val - character to be sent
 * @param value sent to UART3
 * @return none
 */
void putu_3(char val)
{
    while(U3STAbits.UTXBF);
    U3TXREG = val;
}

/*!--------------------------------------------------------------------------------
 * @fn serial_3_init()
 * @breif Initialize serial interface: UART3
 * @param rate - number used to set the baud rate of UART3
 * @return none
 */
void serial_3_init(unsigned long rate)
{
    U3MODEbits.ON = 0;
    U3BRG = (80000000u/(4*rate)) - 1;
    U3MODEbits.BRGH = 1;
    U3MODEbits.ON = 1;
    U3MODEbits.PDSEL = 0;       //8 bit data, no parity
    U3MODEbits.STSEL = 0;       //1 stop bit
    U3MODEbits.UEN = 0b00;      //UxTX and UxRX are enabled and used; no flow
                                 control pins
    U3MODEbits.UARTEN = 1;      //enable UART RX/TX
    U3STAbits.UTXEN = 1;
    U3STAbits.URXEN = 1;
}

/*!--------------------------------------------------------------------------------
 * @fn print_string()
 * @breif Read a given string and display it on the UART1 port using the putu
 *        function in SDlib16.
 * @param string - pointer to the first byte of the string to display
 * @return none
 */
void print_string(char *string)
{
    int length = strlen(string);
    int i;
    for(i = 0; i < length; i++)
    {
        putu(string[i]);
    }
    //putu('\n'); putu('\r');
}

/*!-------------------------------------------------------------------------------
 * @fn print_string2()
 * @breif Read a given string and display it on the UART2 port using the putu2
 *        function in SDlib16.
 * @param string - pointer to the first byte of the string to display
 * @return none
 */
void print_string2(char *string)
{
    int length;
    length = strlen(string);
    int i;
    for(i = 0; i < length; i++)
    {
        putu2(string[i]);
    }
    //putu2('\n'); putu2('\r');
}

/*!-------------------------------------------------------------------------------
 * @fn print_string3()
 * @breif Read a given string and display it on the UART3 port using the putu_3
 *        function in functions17.
 * @param string - pointer to the first byte of the string to display
 * @return none
 */
void print_string3(char *string)
{
    int length;
    length = strlen(string);
    int i;
    for(i = 0; i < length; i++)
    {
        putu_3(string[i]);
    }
    //putu_3('\n'); putu_3('\r');
}

/*!-------------------------------------------------------------------------------
 * @fn configDIP
 * @breif Set bits high to input the value of the DIP switches
 * @param none
 * @return none
 */
void configDIP(void)
{
    TRISEbits.TRISE1 = 1;
    TRISEbits.TRISE2 = 1;
    TRISEbits.TRISE3 = 1;
    TRISEbits.TRISE4 = 1;
    TRISEbits.TRISE5 = 1;
    TRISEbits.TRISE6 = 1;
    TRISEbits.TRISE7 = 1;
    TRISGbits.TRISG6 = 1;
}

uint8 num;
char init_str[20] = {0};
char resp_str[20] = {0};
/*!-------------------------------------------------------------------------------
 * @fn determine_parameters()
 * @breif Use the first 3 DIP switch inputs to determine and display if the
 *        device is an initializer or responder and what it's response number is.
 * @param switch1 - input of the first dip switch
 * @param switch2 - input of the second dip switch
 * @param switch3 - input of the third dip switch
 * @return num - device number
 */
uint8 determine_parameters(uint8 switch1, uint8 switch2, uint8 switch3)
{
    if (switch1 == 0)
    {
        char boardtype[] = "Initalizer";
        if(switch3 == 0){num = 1;} //This is initializer 1
        else{num = 2;} // This is initializer 2
#ifdef DEBUG
        sprintf(init_str, "This is %s %d", boardtype, num);
        print_string(init_str);
        putu('\n'); putu('\r');
#endif
    }
    else
    {
        char boardtype[] = "Responder";
        if(switch2 == 0 && switch3 == 0){num = 1;} //This is responder 1
        else if(switch2 == 0 && switch3 == 1){num = 2;} //This is responder 2
        else if(switch2 == 1 && switch3 == 0){num = 3;} //This is responder 3
        else{num = 4;} //This is responder 4
#ifdef DEBUG
        sprintf(resp_str, "This is %s %d", boardtype, num);
        print_string(resp_str);
        putu('\n'); putu('\r');
#endif
    }
    return num;
}
```

initiator.c

```
/*
 * @file initiator.c
 * @author Katherine Sanders
 *         Eddie Hunckler
 *         Stephen McAndrew
 * @brief Functions to operate the RFPS board as an initiator
 *
 * edited 2.5.17
 */

/* NOTE: The initializer function with the get_distance functions, the
 *       resp_msg_get_ts function, and all their components/ inputs are built on
 *       DecaWave ex_06a_ss_twr_init example.
 *       @brief Single-sided two-way ranging (SS TWR) initiator example code
 * Copyright 2015 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#include "deca_device_api.h"
#include "deca_regs.h"
#include "functions17.h"
#include "magnetometer.h"
#include "port.h"
#include "SDlib16.h"
#include "sleep.h"

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/attribs.h>


//#define DEBUG

/* Frames used in the ranging process. */
uint8 tx_poll_msg1[] = {0x41, 0x88, 0, 0xCA, 0xDE, '1', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 rx_resp_msg1[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '1', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 tx_poll_msg2[] = {0x41, 0x88, 0, 0xCA, 0xDE, '2', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 rx_resp_msg2[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '2', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 tx_poll_msg3[] = {0x41, 0x88, 0, 0xCA, 0xDE, '3', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 rx_resp_msg3[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '3', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 tx_poll_msg4[] = {0x41, 0x88, 0, 0xCA, 0xDE, '4', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 rx_resp_msg4[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '4', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```
/* String used to display measured distance on screen (16 characters maximum). */
char dist_str1[16] = {0}; //””
char dist_str2[16] = {0}; //””
char dist_str3[16] = {0}; //””
char dist_str4[16] = {0}; //””

/* Default communication configuration. We use here EVK1000's mode 4. */
static dwt_config_t config =
{
    2,               // Channel number.
    DWT_PRF_64M,     // Pulse repetition frequency.
    DWT_PLEN_128,    // Preamble length. Used in TX only.
    DWT_PAC8,        // Preamble acquisition chunk size. Used in RX only.
    9,               // TX preamble code. Used in TX only.
    9,               // RX preamble code. Used in RX only.
    0,               // 0 to use standard SFD, 1 to use non-standard SFD.
    DWT_BR_6M8,      // Data rate.
    DWT_PHRMODE_STD, // PHY header mode.
    (129 + 8 - 8)    // SFD timeout (preamble length + 1 + SFD length - PAC
                      size). Used in RX only.
};

/* Frame sequence number, incremented after each transmission. */
static uint8 frame_seq_nb = 0;
/* Hold copy of status register state here for reference so that it can be
 * examined at a debug breakpoint. */
static uint32 status_reg = 0;

uint8 rx_buffer[RX_BUF_LEN];

/* Hold copies of computed time of flight and distance here for reference so that
 * they can be examined at a debug breakpoint. */
static double tof;
static double distance1;
static double distance2;
static double distance3;
static double distance4;
uint16 int_distance1;
uint16 int_distance2;
uint16 int_distance3;
uint16 int_distance4;
uint16 avg_distance1[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint16 avg_distance2[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint16 avg_distance3[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint16 avg_distance4[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint16 avg_dist1;
uint16 avg_dist2;
uint16 avg_dist3;
uint16 avg_dist4;
char command; //This is the serial input
uint8 failcount1 = 0;
uint8 failcount2 = 0;
uint8 failcount3 = 0;
```

```
uint8 failcount4 = 0;

int x_vector;
int y_vector;
int z_vector;
char x_string[10] = "";
char y_string[10] = "";
char z_string[10] = "";


/* Declaration of static functions. */
static void resp_msg_get_ts(uint8 *ts_field, uint32 *ts);
static uint16 find_average(uint16 array[]);
static uint16 get_distance1(void);
static uint16 get_distance2(void);
static uint16 get_distance3(void);
static uint16 get_distance4(void);
static void send_wifi_data(uint16 int_distance1, uint16 int_distance2, uint16
                           int_distance3, uint16 int_distance4);
static void send_serial_data(uint16 *avg_distance1, uint16 *avg_distance2, uint16
                             *avg_distance3, uint16 *avg_distance4);


/*! ----------------------------------------------------------------------------
 * @fn initializer()
 * @brief Setup board as an initializer communicating with four responders using
 *        preset messages. Send distances to each responder to UART3 and UART1.
 * @param enableWifi - input of the eighth DIP switch
 * @return none
 */
void initializer(int enableWifi)
{
    /* Start with board specific hardware initialization. */
    peripherals_init();

#ifdef DEBUG
    /* Initialize toggle bits for debug */
    TRIS_D1 = 0;
    toggle_d1
    /* Display application name. */
    print_string("INITIATOR");
    putu('\n'); putu('\r');
#endif

    /* Reset and initialize DW1000.
     * For initialization, DW1000 clocks must be temporarily set to crystal speed.
     * After initialization SPI rate can be increased for optimum performance. */
    reset_DW1000(); // Target specific drive of RSTn line into DW1000 low for a
                    period.
    spi_set_rate_low();
    if (dwt_initialise(DWT_LOADUCODE) == DWT_ERROR)
    {
#ifdef DEBUG
```

```
    print_string("INIT FAILED");
    putu('\n'); putu('\r');
#endif
        while (1)
        { };
    }
    spi_set_rate_high();
    /* Configure DW1000. */
    dwt_configure(&config);
    /* Apply default antenna delay value. */
    dwt_setrxantennadelay(RX_ANT_DLY);
    dwt_settxantennadelay(TX_ANT_DLY);
    /* Set expected response's delay and timeout.
     * As this only handles one incoming frame with always the same delay and
     * timeout, those values can be set here once for all. */
    dwt_setrxaftertxdelay(POLL_TX_TO_RESP_RX_DLY_UUS);
    dwt_setrxtimeout(RESP_RX_TIMEOUT_UUS);

    /* Loop forever initiating ranging exchanges. */
    while (1)
    {
        get_distance1();
        get_distance2();
        get_distance3();
        get_distance4();

        int i;
        if(int_distance1 != 0)
        {
            for(i = 10; i--; i > 0)
            {
                avg_distance1[i] = avg_distance1[i-1];
            }
            avg_distance1[0] = int_distance1;
        }
        if(int_distance2 != 0)
        {
            for(i = 10; i--; i > 0)
            {
                avg_distance2[i] = avg_distance2[i-1];
            }
            avg_distance2[0] = int_distance2;
        }
        if(int_distance3 != 0)
        {
            for(i = 10; i--; i > 0)
            {
                avg_distance3[i] = avg_distance3[i-1];
            }
            avg_distance3[0] = int_distance3;
        }
        if(int_distance4 != 0)
        {
            for(i = 10; i--; i > 0)
            {
                avg_distance4[i] = avg_distance4[i-1];
            }
            avg_distance4[0] = int_distance4;
        }

        if(enableWifi)
        {
            send_wifi_data(int_distance1, int_distance2, int_distance3,
                           int_distance4);
            if(MEASUREMENT_AVAILABLE)
            {
                /* THESE FUNCTIONS MUST BE CALLED IN THIS ORDER*/
                x_vector = getX();
                y_vector = getY();
                z_vector = getZ();

                sprintf(x_string,"RX<%d>", x_vector);
                print_string3(x_string);
                sprintf(y_string,"RY<%d>", y_vector);
                print_string3(y_string);
                sprintf(z_string,"RZ<%d>", z_vector);
                print_string3(z_string);
            }
        }
        send_serial_data(avg_distance1, avg_distance2, avg_distance3,
                         avg_distance4);

        /* Execute a delay between ranging exchanges. */
        sleep_ms(RNG_DELAY_MS);
    }
}

/*! ---------------------------------------------------------------------------
 * @fn get_distance1()
 * @brief Send and receive a message to and from Responder 1 and calculate
 *        distance to responder
 * @param none
 * @return int_distance1 - the integer distance from initiator to Responder 1
 *                         (0 if transmission error)
 */
static uint16 get_distance1(void)
{
    /* Write frame data to DW1000 and prepare transmission. */
    tx_poll_msg1[ALL_MSG_SN_IDX] = frame_seq_nb;
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
    dwt_writetxdata(sizeof(tx_poll_msg1), tx_poll_msg1, 0); // Zero offset in TX
                                                              buffer.
    dwt_writetxfctrl(sizeof(tx_poll_msg1), 0, 1); // Zero offset in TX buffer,
                                                     ranging.
    /* Start transmission, indicating that a response is expected so that
     * reception is enabled automatically after the frame is sent and the delay
     * set by dwt_setrxaftertxdelay() has elapsed. */
    dwt_starttx(DWT_START_TX_IMMEDIATE | DWT_RESPONSE_EXPECTED);
    /* Assume that the transmission is achieved correctly, poll for reception of
     * a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_TO | SYS_STATUS_ALL_RX_ERR)))
    { };
    /* Increment frame sequence number after transmission of the poll message
     * (modulo 256). */
    frame_seq_nb++;
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFLEN_MASK;
        if (frame_len <= RX_BUF_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is the expected response.
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_resp_msg1, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 poll_tx_ts, resp_rx_ts, poll_rx_ts, resp_tx_ts;
            int32 rtd_init, rtd_resp;
            /* Retrieve poll transmission and response reception timestamps. */
            poll_tx_ts = dwt_readtxtimestamplo32();
            resp_rx_ts = dwt_readrxtimestamplo32();
            /* Get timestamps embedded in response message. */
            resp_msg_get_ts(&rx_buffer[RESP_MSG_POLL_RX_TS_IDX], &poll_rx_ts);
            resp_msg_get_ts(&rx_buffer[RESP_MSG_RESP_TX_TS_IDX], &resp_tx_ts);
            /* Compute time of flight and distance. */
            rtd_init = resp_rx_ts - poll_tx_ts;
            rtd_resp = resp_tx_ts - poll_rx_ts;
            tof = ((rtd_init - rtd_resp) / 2.0) * DWT_TIME_UNITS;
            distance1 = tof * SPEED_OF_LIGHT * METERS_TO_INCHES;
            if(distance1 > 0)
            {
                int_distance1 = (int) distance1;
                failcount1 = 0;
            }

#ifdef DEBUG
            /* Display computed distance. */
            sprintf(dist_str1, "R1<%3.2f inches>", distance1);
            print_string(dist_str1);
#endif

        }
    }
    else
    {
        /* Clear RX error/timeout events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_TO |
                          SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialise LDE operation. */
        dwt_rxreset();

#ifdef DEBUG
        print_string("R1<NOT RECEIVED>");
#endif

        failcount1++;
        if(failcount1 >= ACCEPTABLE_FAILS)
        {
            int_distance1 = 0;
            failcount1 = 0;
        }
    }
    return int_distance1;
}

/*! ------------------------------------------------------------------------------------
 * @fn get_distance2()
 * @brief Send and receive a message to and from Responder 2 and calculate
 *        distance to responder
 * @param none
 * @return int_distance2 - the integer distance from initiator to Responder 2
 *                         (0 if transmission error)
 */
static uint16 get_distance2(void)
{
    /* Write frame data to DW1000 and prepare transmission. */
    tx_poll_msg2[ALL_MSG_SN_IDX] = frame_seq_nb;
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
    dwt_writetxdata(sizeof(tx_poll_msg2), tx_poll_msg1, 0); // Zero offset in TX
                                                              buffer.
    dwt_writetxfctrl(sizeof(tx_poll_msg2), 0, 1); // Zero offset in TX buffer,
                                                    ranging.
    /* Start transmission, indicating that a response is expected so that
     * reception is enabled automatically after the frame is sent and the delay
     * set by dwt_setrxaftertxdelay() has elapsed. */
    dwt_starttx(DWT_START_TX_IMMEDIATE | DWT_RESPONSE_EXPECTED);
    /* Assume that the transmission is achieved correctly, poll for reception of
     * a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_TO | SYS_STATUS_ALL_RX_ERR)))
    { };
    /* Increment frame sequence number after transmission of the poll message
     * (modulo 256). */
    frame_seq_nb++;
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFLEN_MASK;
        if (frame_len <= RX_BUF_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is the expected response.
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_resp_msg2, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 poll_tx_ts, resp_rx_ts, poll_rx_ts, resp_tx_ts;
            int32 rtd_init, rtd_resp;
            /* Retrieve poll transmission and response reception timestamps. */
            poll_tx_ts = dwt_readtxtimestamplo32();
            resp_rx_ts = dwt_readrxtimestamplo32();
            /* Get timestamps embedded in response message. */
            resp_msg_get_ts(&rx_buffer[RESP_MSG_POLL_RX_TS_IDX], &poll_rx_ts);
            resp_msg_get_ts(&rx_buffer[RESP_MSG_RESP_TX_TS_IDX], &resp_tx_ts);
            /* Compute time of flight and distance. */
            rtd_init = resp_rx_ts - poll_tx_ts;
            rtd_resp = resp_tx_ts - poll_rx_ts;
            tof = ((rtd_init - rtd_resp) / 2.0) * DWT_TIME_UNITS;
            distance2 = tof * SPEED_OF_LIGHT * METERS_TO_INCHES;
            if(distance2 > 0)
            {
                int_distance2 = (int) distance2;
                failcount2 = 0;
            }

#ifdef DEBUG
            /* Display computed distance. */
            sprintf(dist_str2, "R2<%3.2f inches>", distance2);
            print_string(dist_str2);
#endif

        }
    }
    else
    {
        /* Clear RX error/timeout events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_TO |
                          SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialise LDE operation. */
        dwt_rxreset();

#ifdef DEBUG
        print_string("R2<NOT RECEIVED>");
```

```
#endif

        failcount2++;
        if(failcount2 >= ACCEPTABLE_FAILS)
        {
            int_distance2 = 0;
            failcount2 = 0;
        }
    }
    return int_distance2;
}

/*! ---------------------------------------------------------------------------
 * @fn get_distance3()
 * @brief Send and receive a message to and from Responder 3 and calculate
 *        distance to responder
 * @param none
 * @return int_distance3 - the integer distance from initiator to Responder 3
 *                         (0 if transmission error)
 */
static uint16 get_distance3(void)
{
    /* Write frame data to DW1000 and prepare transmission. */
    tx_poll_msg3[ALL_MSG_SN_IDX] = frame_seq_nb;
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
    dwt_writetxdata(sizeof(tx_poll_msg3), tx_poll_msg3, 0); // Zero offset in TX
                                                              buffer.
    dwt_writetxfctrl(sizeof(tx_poll_msg3), 0, 1); // Zero offset in TX buffer,
                                                   ranging.
    /* Start transmission, indicating that a response is expected so that
     * reception is enabled automatically after the frame is sent and the delay
     * set by dwt_setrxaftertxdelay() has elapsed. */
    dwt_starttx(DWT_START_TX_IMMEDIATE | DWT_RESPONSE_EXPECTED);
    /* Assume that the transmission is achieved correctly, poll for reception of
     * a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_TO | SYS_STATUS_ALL_RX_ERR)))
    { };
    /* Increment frame sequence number after transmission of the poll message
     * (modulo 256). */
    frame_seq_nb++;
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFLEN_MASK;
        if (frame_len <= RX_BUF_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is the expected response.
```

```
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_resp_msg3, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 poll_tx_ts, resp_rx_ts, poll_rx_ts, resp_tx_ts;
            int32 rtd_init, rtd_resp;
            /* Retrieve poll transmission and response reception timestamps. */
            poll_tx_ts = dwt_readtxtimestamplo32();
            resp_rx_ts = dwt_readrxtimestamplo32();
            /* Get timestamps embedded in response message. */
            resp_msg_get_ts(&rx_buffer[RESP_MSG_POLL_RX_TS_IDX], &poll_rx_ts);
            resp_msg_get_ts(&rx_buffer[RESP_MSG_RESP_TX_TS_IDX], &resp_tx_ts);
            /* Compute time of flight and distance. */
            rtd_init = resp_rx_ts - poll_tx_ts;
            rtd_resp = resp_tx_ts - poll_rx_ts;
            tof = ((rtd_init - rtd_resp) / 2.0) * DWT_TIME_UNITS;
            distance3 = tof * SPEED_OF_LIGHT * METERS_TO_INCHES;
            if(distance3 > 0)
            {
                int_distance3 = (int) distance3;
                failcount3 = 0;
            }

#ifdef DEBUG
            /* Display computed distance. */
            sprintf(dist_str3, "R3<%3.2f inches>", distance3);
            print_string(dist_str3);
#endif

        }
    }
    else
    {
        /* Clear RX error/timeout events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_TO |
                          SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialise LDE operation. */
        dwt_rxreset();

#ifdef DEBUG
        print_string("R3<NOT RECEIVED>");
#endif

        failcount3++;
        if(failcount3 >= ACCEPTABLE_FAILS)
        {
            int_distance3 = 0;
            failcount3 = 0;
        }
    }
    return int_distance3;
}
```

```
/*! ------------------------------------------------------------------------
 * @fn get_distance4()
 * @brief Send and receive a message to and from Responder 4 and calculate
 *        distance to responder
 * @param none
 * @return int_distance4 - the integer distance from initiator to Responder 4
 *                         (0 if transmission error)
 */
static uint16 get_distance4(void)
{
    /* Write frame data to DW1000 and prepare transmission. */
    tx_poll_msg4[ALL_MSG_SN_IDX] = frame_seq_nb;
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
    dwt_writetxdata(sizeof(tx_poll_msg4), tx_poll_msg4, 0); // Zero offset in TX
                                                              buffer.
    dwt_writetxfctrl(sizeof(tx_poll_msg4), 0, 1); // Zero offset in TX buffer,
                                                    ranging.
    /* Start transmission, indicating that a response is expected so that
     * reception is enabled automatically after the frame is sent and the delay
     * set by dwt_setrxaftertxdelay() has elapsed. */
    dwt_starttx(DWT_START_TX_IMMEDIATE | DWT_RESPONSE_EXPECTED);
    /* Assume that the transmission is achieved correctly, poll for reception of
     * a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_TO | SYS_STATUS_ALL_RX_ERR)))
    { };
    /* Increment frame sequence number after transmission of the poll message
     * (modulo 256). */
    frame_seq_nb++;
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFLEN_MASK;
        if (frame_len <= RX_BUF_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is the expected response.
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_resp_msg4, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 poll_tx_ts, resp_rx_ts, poll_rx_ts, resp_tx_ts;
            int32 rtd_init, rtd_resp;
            /* Retrieve poll transmission and response reception timestamps. */
            poll_tx_ts = dwt_readtxtimestamplo32();
            resp_rx_ts = dwt_readrxtimestamplo32();
            /* Get timestamps embedded in response message. */
            resp_msg_get_ts(&rx_buffer[RESP_MSG_POLL_RX_TS_IDX], &poll_rx_ts);
            resp_msg_get_ts(&rx_buffer[RESP_MSG_RESP_TX_TS_IDX], &resp_tx_ts);
            /* Compute time of flight and distance. */
            rtd_init = resp_rx_ts - poll_tx_ts;
            rtd_resp = resp_tx_ts - poll_rx_ts;
            tof = ((rtd_init - rtd_resp) / 2.0) * DWT_TIME_UNITS;
            distance4 = tof * SPEED_OF_LIGHT * METERS_TO_INCHES;
            if(distance4 > 0)
            {
                int_distance4 = (int) distance4;
                failcount4 = 0;
            }

#ifdef DEBUG
            /* Display computed distance. */
            sprintf(dist_str4, "R4<%3.2f inches>", distance4);
            print_string(dist_str4);
#endif

        }
    }
    else
    {
        /* Clear RX error/timeout events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_TO |
                          SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialize LDE operation. */
        dwt_rxreset();

#ifdef DEBUG
        print_string("R4<NOT RECEIVED>");
#endif

        failcount4++;
        if(failcount4 >= ACCEPTABLE_FAILS)
        {
            int_distance4 = 0;
            failcount4 = 0;
        }
    }
    return int_distance4;
}

/*! ---------------------------------------------------------------------------
 * @fn resp_msg_get_ts()
 * @brief Read a given timestamp value from the response message. In the
 *        timestamp fields of the response message, the least significant byte is
 *        at the lower address.
 * @param ts_field - pointer on the first byte of the timestamp field to get
 * @param ts - timestamp value
 * @return none
 */
static void resp_msg_get_ts(uint8 *ts_field, uint32 *ts)
```

```
{
    int i;
    *ts = 0;
    for (i = 0; i < RESP_MSG_TS_LEN; i++)
    {
        *ts += ts_field[i]<<(i * 8);
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn find_average()
 * @brief Given an array of 10 numbers, find the average value of those numbers
 * @param array - an array of 10 numbers
 * @return average - average value of the given array
 */
static uint16 find_average (uint16 array[])
{
    int k = 0;
    uint16 sum = 0;
    uint16 average = 0;
    for (k = 0; k < 10; k++)
    {
        sum += array[k];
    }
    average = sum / 10;
    return average;
}

/*! ------------------------------------------------------------------------------------
 * @fn send_wifi_data()
 * @brief Send the measured distance (from all responders) to UART3
 * @param int_distance1 - most recent distance to Responder 1
 * @param int_distance2 - most recent distance to Responder 2
 * @param int_distance3 - most recent distance to Responder 3
 * @param int_distance4 - most recent distance to Responder 4
 * @param value sent to MQTT server
 * @return none
 */
static void send_wifi_data (uint16 int_distance1, uint16 int_distance2, uint16
                            int_distance3, uint16 int_distance4)
{
    sprintf(dist_str1, "R1<%d>", int_distance1);
    print_string3(dist_str1);
    sprintf(dist_str2, "R2<%d>", int_distance2);
    print_string3(dist_str2);
    sprintf(dist_str3, "R3<%d>", int_distance3);
    print_string3(dist_str3);
    sprintf(dist_str4, "R4<%d>", int_distance4);
    print_string3(dist_str4);
}

/*! ------------------------------------------------------------------------------------
 * @fn send_serial_data()
 * @brief When given a command from UART1, calculate the average distance for
 *        that responder and send to UART1
 * @param avg_distance1 - array of last ten distances to Responder 1
 * @param avg_distance2 - array of last ten distances to Responder 2
 * @param avg_distance3 - array of last ten distances to Responder 3
 * @param avg_distance4 - array of last ten distances to Responder 4
 * @param value sent to display via UART1.
 * @return none
 */
static void send_serial_data (uint16 *avg_distance1, uint16 *avg_distance2,
                              uint16 *avg_distance3, uint16 *avg_distance4)
{
    if(U1STAbits.URXDA)
    {
        command = getu();
        switch(command)
        {
            case '1':
                avg_dist1 = find_average(avg_distance1);
                putu(((0xFF00&avg_dist1)>>8) & 0xFF);
                putu(0x00FF&avg_dist1);
                break;
            case '2':
                avg_dist2 = find_average(avg_distance2);
                putu(((0xFF00&avg_dist2)>>8) & 0xFF);
                putu(0x00FF&avg_dist2);
                break;
            case '3':
                avg_dist3 = find_average(avg_distance3);
                putu(((0xFF00&avg_dist3)>>8) & 0xFF);
                putu(0x00FF&avg_dist3);
                break;
            case '4':
                avg_dist4 = find_average(avg_distance4);
                putu(((0xFF00&avg_dist4)>>8) & 0xFF);
                putu(0x00FF&avg_dist4);
                break;
            default:
                break;
        }
    }
}
```

lcd.c

```
/*! ----------------------------------------------------------------------------
 * @file    lcd.c
 * @brief   EVB1000 LCD screen access functions
 * Copyright 2015 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 22.2.17
 */
```

```c
//#include "sleep.h"
//#include "port.h"
#include "lcd.h"
#include "SDlib16.h"

#include <xc.h>
#include <string.h>


/*! --------------------------------------------------------------------------------
 * @fn writetoLCD()
 * @brief Write data to the LCD display via SPI2.
 * @param bodylength - length of the input buffer
 * @param rs_enable – enable for reset
 * @param bodyBuffer - buffer with characters to be displayed
 * @param characters sent to/ displayed on LCD screen
 * @return none
 */
void writetoLCD
(
    int bodylength,
    int rs_enable,
    const char *bodyBuffer
)
{
    /* Return cursor home and clear screen. */
    LCD_clear();
    LCD_setpos(0x00,0x00);
    int i;
    for(i = 0; i < bodylength; i++)
    {
        LCD_char(bodyBuffer[i]); //send data on the SPI
    }
}

/*! --------------------------------------------------------------------------------
 * @fn printHEX()
 * @brief Write hex value to the LCD display via SPI2.
 * @param num - number to be displayed in hex
 * @param value displayed on LCD screen
 * @return none
 */
void printHEX(unsigned long num)
{
    LCD_clear();
    LCD_setpos(0x00,0x00);

    int digit7 = (int)(num/268435456) %16;
    char digit7c;
    if (digit7<10){digit7c = digit7 + 48;}
    else{digit7c = digit7 + 55;}
    int digit6 = (int)(num/16777216) %16;
    char digit6c;
    if (digit6<10){digit6c = digit6 + 48;}
    else{digit6c = digit6 + 55;}
    int digit5 = (int)(num/1048576) %16;
    char digit5c;
    if (digit5<10){digit5c = digit5 + 48;}
    else{digit5c = digit5 + 55;}
    int digit4 = (int)(num/65536) %16;
    char digit4c;
    if (digit4<10){digit4c = digit4 + 48;}
    else{digit4c = digit4 + 55;}
    int digit3 = (int)(num/4096) %16;
    char digit3c;
    if (digit3<10){digit3c = digit3 + 48;}
    else{digit3c = digit3 + 55;}
    int digit2 = (int)(num/256) %16;
    char digit2c;
    if (digit2<10){digit2c = digit2 + 48;}
    else{digit2c = digit2 + 55;}
    int digit1 = (int)(num/16) %16;
    char digit1c;
    if (digit1<10){digit1c = digit1 + 48;}
    else{digit1c = digit1 + 55;}
    int digit0 = (int)(num/1) %16;
    char digit0c;
    if (digit0<10){digit0c = digit0 + 48;}
    else{digit0c = digit0 + 55;}

    LCD_char('0');
    LCD_char('x');
    LCD_char(digit7c);
    LCD_char(digit6c);
    LCD_char(digit5c);
    LCD_char(digit4c);
    LCD_char(digit3c);
    LCD_char(digit2c);
    LCD_char(digit1c);
    LCD_char(digit0c);
}

/*! -------------------------------------------------------------------------------
 * @fn lcd_display_str()
 * @brief Display a string on the LCD screen.
 *        NOTE: The string must be 16 chars long maximum!
 * @param string - the string to display
 * @return none
 */
void lcd_display_str(const char *string)
{
    /* Write the string to display. */
    writetoLCD(strlen(string), 1, (const char *)string);
}
```

magnetometer.c

```
/*
 * @file magnetometer.c
 * @author Stephen McAndrew
 *         Eddie Hunckler
 * @brief Functions to operate the magnetometer on the RFPS board
 *
 * edited 26.4.17
 */

#include "magnetometer.h"


/*! -----------------------------------------------------------------------------
 * @fn getX()
 * @brief Get value for X directionality
 * @param none
 * @return x_data_int - integer value for directionality
 */
int getX(void)
{
    I2C_start(); // send start condition
    I2C_write(WRITE_DEVICE); // device address with write bit set
    I2C_write(OUT_X_MSB); // address of the 1st data register (x MSB): must be
                           read to clear RD11 pin
    I2C_restart(); // send re-start condition
    I2C_write(READ_DEVICE); // device address with read bit clear
    MSB_x = (unsigned int)I2C_read(NACK); // read 0x01 with NACK (0x00)
    LSB_x = (unsigned int)I2C_read(ACK); // read 0x02 with NACK (0x00)
    I2C_stop();
    /* shift MSB */
    x_data = (((MSB_x<<8) & 0xFF00)| LSB_x);
    /* 2's Complement */
    if(x_data > 32767)
    {
        x_data_int = x_data - 65536;
    }
    else
    {
        x_data_int = x_data;
    }
    return(x_data_int);
}

/*! ----------------------------------------------------------------------------
 * @fn getY()
 * @brief Get value for Y directionality
 * @param none
 * @return y_data_int - integer value for directionality
 */
int getY(void)
{
    I2C_start(); // send start condition
    I2C_write(WRITE_DEVICE); // device address with write bit set
    I2C_write(OUT_Y_MSB); // address of the 1st data register (x MSB): must be
                           read to clear RD11 pin
    I2C_restart(); // send re-start condition
    I2C_write(READ_DEVICE); // device address with read bit clear
    MSB_y = (unsigned int)I2C_read(NACK); // read 0x01 with NACK (0x00)
    LSB_y = (unsigned int)I2C_read(ACK); // read 0x02 with NACK (0x00)
    I2C_stop();
    y_data = (((MSB_y<<8) & 0xFF00)| LSB_y);
    /* 2's Complement */
    if(y_data > 32767)
    {
        y_data_int = y_data - 65536;
    }
    else
    {
        y_data_int = y_data;
    }
    return(y_data_int);
}

/*! --------------------------------------------------------------------------------
 * @fn getZ()
 * @brief Get value for Z directionality
 * @param none
 * @return z_data_int - integer value for directionality
 */
int getZ(void)
{
    I2C_start(); // send start condition
    I2C_write(WRITE_DEVICE); // device address with write bit set
    I2C_write(OUT_Z_MSB); // address of the 1st data register (x MSB): must be
                           read to clear RD11 pin
    I2C_restart(); // send re-start condition
    I2C_write(READ_DEVICE); // device address with read bit clear
    MSB_z = (unsigned int)I2C_read(NACK); // read 0x01 with NACK (0x00)
    LSB_z = (unsigned int)I2C_read(ACK); // read 0x02 with NACK (0x00)
    I2C_stop();
    z_data = (((MSB_z<<8) & 0xFF00)| LSB_z);
    /* 2's Complement */
    if(z_data > 32767)
    {
        z_data_int = z_data - 65536;
    }
    else
    {
        z_data_int = z_data;
    }
    return(z_data_int);
}

/*! --------------------------------------------------------------------------------
```

```
 * @fn MAG3110_init()
 * @brief Initialize the magnetometer
 * @param none
 * @param set control registers for I2C
 * @return none
 */
void MAG3110_init(void)
{
    I2C_init(100000UL);
   /* configure CTRL_REG2 */
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(CTRL_REG2);             // CTRL_REG2 address
   I2C_write(0x80);                  // set the automatic reset bit in CTRL_REG2
   I2C_stop();                       // sens stop condition

   /* configure CTRL_REG1 */
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(CTRL_REG1);             // CTRL_REG1 address
   I2C_write(0xC9);                  // set ODR = 0.63Hz OSR = 2
   I2C_stop();                       // send stop condition

   /* set x offset */
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_X_MSB);             // x MSB offset address
   I2C_write((XOFFSET&(0xFF00))>>8); // send the value of MSB the offset
   I2C_stop();                       // send stop condition
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_X_LSB);             // x LSB offset address
   I2C_write(XOFFSET&(0x00FF));      // send the value of MSB the offset
   I2C_stop();                       // send stop condition

   /* set y offset */
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_Y_MSB);             // y MSB offset address
   I2C_write((YOFFSET&(0xFF00))>>8); // send the value of MSB the offset
   I2C_stop();                       // send stop condition
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_Y_LSB);             // y LSB offset address
   I2C_write(YOFFSET&(0x00FF));      // send the value of MSB the offset
   I2C_stop();                       // send stop condition

   /* set z offset */
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_Z_MSB);             // z MSB offset address
   I2C_write((ZOFFSET&(0xFF00))>>8); // send the value of MSB the offset
   I2C_stop();                       // send stop condition
   I2C_start();                      // send start condition
   I2C_write(WRITE_DEVICE);          // device address with write bit set
   I2C_write(OFF_Z_LSB);             // z LSB offset address
   I2C_write(ZOFFSET&(0x00FF));      // send the value of MSB the offset
   I2C_stop();                       // send stop condition
}

/*! ------------------------------------------------------------------------------
 * @fn I2C_init()
 * @brief Initialize I2C
 * @param rate - number used to set the baud rate of I2C
 * @return 1 - success
 */
unsigned char I2C_init(unsigned long rate)
{
    AD1PCFG = 0xFFFF;                     //set all pins to digital
    DDPCONbits.JTAGEN = 0;                //disable JTAG
    I2C1CONbits.ON = 0;                   //turn off the module
    I2C1BRG = get_pb_clock()/2/rate-get_pb_clock()/10000000UL-1; //set BRG rate
    I2C1CONbits.ON = 1;                   //turn on the module
    return 1;
}

/*! ------------------------------------------------------------------------------
 * @fn I2C_start()
 * @brief Start I2C
 * @param none
 * @return 1 - success
 */
unsigned char I2C_start(void)
{

    IEC0bits.I2C1MIE = 1;          //enable interrupt
    IFS0bits.I2C1MIF = 0;          //clear interrupt flag
    I2C1CONbits.SEN = 1;           //start condition enable bit
    while(IFS0bits.I2C1MIF != 1); //wait for interrupt flag
    IFS0bits.I2C1MIF = 0;          //clear interrupt flag
    return 1;
}

/*! ------------------------------------------------------------------------------
 * @fn I2C_stop()
 * @brief Stop I2C
 * @param none
 * @return 1 - success
 */
unsigned char I2C_stop(void)
{
   IEC0bits.I2C1MIE = 1;          //enable interrupt
   IFS0bits.I2C1MIF = 0;          //clear interrupt flag
   I2C1CONbits.PEN = 1;           //stop condition enable bit
   while(IFS0bits.I2C1MIF != 1); //wait for interrupt flag
   IFS0bits.I2C1MIF = 0;          //clear interrupt flag
   return 1;
}

/*! --------------------------------------------------------------------------------
 * @fn I2C_restart()
 * @brief Restart I2C
 * @param none
 * @return 1 - success
 */
unsigned char I2C_restart(void)
{
   IEC0bits.I2C1MIE = 1;         //enable interrupt
   IFS0bits.I2C1MIF = 0;         //clear interrupt flag
   I2C1CONbits.RSEN = 1;         //restart condition enable bit
   while(IFS0bits.I2C1MIF != 1); //wait for interrupt flag
   IFS0bits.I2C1MIF = 0;         //clear interrupt flag
   return 1;
}

/*! --------------------------------------------------------------------------------
 * @fn I2C_write()
 * @brief Write to I2C
 * @param data – what is written
 * @return I2C1STATbits.ACKSTAT - note that ack received
 */
char I2C_write(char data)
{
    IEC0bits.I2C1MIE = 1;         //enable interrupt
    IFS0bits.I2C1MIF = 0;         //clear interrupt flag
    I2C1TRN = data;               //load transmit buffer
    while(IFS0bits.I2C1MIF != 1); //wait for interrupt flag
    IFS0bits.I2C1MIF = 0;         //clear interrupt flag
    return I2C1STATbits.ACKSTAT;  //return ack received
}

/*! --------------------------------------------------------------------------------
 * @fn I2C_read()
 * @brief Read with I2C
 * @param ack – what is read
 * @return I2C1RCV - data read
 */
unsigned char I2C_read(unsigned char ack)
{
    IEC0bits.I2C1MIE = 1;         //enable interrupt
    IFS0bits.I2C1MIF = 0;         //clear interrupt flag
    I2C1CONbits.RCEN = 1;         // set receive enable bit
    while(IFS0bits.I2C1MIF != 1); //wait for interrupt flag
    IFS0bits.I2C1MIF = 0;         //clear interrupt flag
    I2C1CONbits.ACKDT = ack;      //initiate ack event
    I2C1CONbits.ACKEN = 1;
    while(IFS0bits.I2C1MIF != 1);
    IFS0bits.I2C1MIF = 0;         //clear interrupt flag
    return I2C1RCV;               //return the data read
```

```
}
```

port.c

```
/*! ------------------------------------------------------------------------------
 * @file port.c
 * @brief HW specific definitions and functions for portability
 * Copyright 2013 (c) DecaWave Ltd, Dublin, Ireland.
 * All rights reserved.
 *
 * edited 25.4.17
 */

#include "functions17.h"
#include "lcd.h"
#include "port.h"
#include "SDlib16.h"
#include "sleep.h"

#include <xc.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>


//#define interrupt_init(x)                         NVIC_Configuration(x)
//#define usart1_init(x)                            USART1_Configuration(x)
#define spi_init(x)                               SPI_Configuration(x)
//#define gpio_init(x)                              GPIO_Configuration(x)
#define lcd_init(x)                               LCD_Configuration(x)

//#define USART_SUPPORT

/* Declaration of static functions. */
static int USART1_Configuration(void);
static void LCD_Configuration(void);
static void spi_peripheral_init(void);

/*! -------------------------------------------------------------------------------
 * @fn USART1_Configuration()
 * @brief Configure USART1.
 * @param none
 * @return zero
 */
static int USART1_Configuration(void)
{
    serial_init(115200u); //Function from SDlib16.
    return 0;
}

/*! -------------------------------------------------------------------------------
 * @fn spi_set_rate_low()
 * @brief Set SPI rate to less than 3 MHz to properly perform DW1000
 *        initialisation.
 * @param none
 * @return none
 */
void spi_set_rate_low (void)
{
    SPI4CONbits.ON = 0;
    SPI4BRG = 39; //SPI clock of 1MHz
    SPI4CONbits.ON = 1;
}

/*! ------------------------------------------------------------------------------------
 * @fn spi_set_rate_high()
 * @brief Set SPI rate as close to 20 MHz as possible for optimum performances.
 * @param none
 * @return none
 */
void spi_set_rate_high (void)
{
    SPI4CONbits.ON = 0;
    SPI4BRG = 7; //SPI clock of 5MHz
    SPI4CONbits.ON = 1;
}

/*! ------------------------------------------------------------------------------------
 * @fn SPI_Configuration()
 * @brief Configure SPI4.
 * @param none
 * @return zero
 */
int SPI_Configuration(void)
{
    //Setup Pins
    TRISBbits.TRISB8 = 0;
    AD1PCFG = 0xFFFF; //Turns off analog inputs
    DDPCONbits.JTAGEN = 0; //Turns off JTAG
    //Initial State
    SSLAT = 1;
    //Setup SPI
    SPI4CONbits.ON = 0;
    SPI4BRG = 19;
    SPI4CONbits.SIDL = 0;
    SPI4CONbits.MODE32 = 0;
    SPI4CONbits.MODE16 = 0;
    SPI4CONbits.SMP = 0;
    SPI4CONbits.SSEN = 0;
    SPI4CONbits.CKE = 1;
    SPI4CONbits.CKP = 0;
    SPI4CONbits.MSTEN = 1;
    SPI4CONbits.DISSDO = 0;
    SPI4CONbits.STXISEL = 0b00;
    SPI4CONbits.SRXISEL = 0b00;
    //SPI4STATbits.SPIROV = 0;
    SPI4CONbits.ON = 1;
    return 0;
}

/*! ------------------------------------------------------------------------------------
 * @fn reset_DW1000()
 * @brief Reset the DW1000 chip using PIC bits.
 * @param none
 * @return none
 */
void reset_DW1000(void)
{
    RESETTRIS = 0; // Enable GPIO used for DW1000 reset
    RESETLAT = 0;  //drive the RSTn pin low
    RESETTRIS = 1; //put the pin back to tri-state ... as input
    sleep_ms(2);
}

/*! ------------------------------------------------------------------------------------
 * @fn LCD_configuration()
 * @brief Initialize LCD screen.
 * @param none
 * @return none
 */
static void LCD_Configuration(void)
{
    LCD_init(); //Function from SDlib16
}

/*! ------------------------------------------------------------------------------------
 * @fn spi_peripheral_init()
 * @brief Initialize all SPI peripherals at once.
 * @param none
 * @return none
*/
static void spi_peripheral_init(void)
{
    spi_init();
    // Wait for LCD to power on.
    sleep_ms(10);
}

/*! ------------------------------------------------------------------------------------
 * @fn peripherals_init()
 * @brief Initialize all peripherals.
 * @param none
 * @return none
 */
void peripherals_init (void)
{
     //gpio_init();
     spi_peripheral_init();
     //lcd_init();
```

```
//#ifdef USART_SUPPORT
    //usart1_init();
//#endif

}
```

---

responder.c

```
/*
 * @file responder.c
 * @author Katherine Sanders
 *         Eddie Hunckler
 *         Stephen McAndrew
 * @brief Functions to operate the RFPS board as a responder
 *
 * edited 2.5.17
 */

/* NOTE: The responder + get_distance functions, get_rx_timestamp_u64 function,
 *       resp_msg_set_ts function, and their components/ inputs are built on
 *       DecaWave ex_06b_ss_twr_resp example.
 *       @brief   Single-sided two-way ranging (SS TWR) responder example code
 * Copyright 2015 (c) Decawave Ltd, Dublin, Ireland.
 * All rights reserved.
 */

#include "deca_device_api.h"
#include "deca_regs.h"
#include "functions17.h"
#include "port.h"
#include "SDlib16.h"
#include "sleep.h"

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//#define DEBUG

/* Frames used in the ranging process. */
uint8 rx_poll_msg1[] = {0x41, 0x88, 0, 0xCA, 0xDE, '1', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 tx_resp_msg1[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '1', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 rx_poll_msg2[] = {0x41, 0x88, 0, 0xCA, 0xDE, '2', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 tx_resp_msg2[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '2', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 rx_poll_msg3[] = {0x41, 0x88, 0, 0xCA, 0xDE, '3', 'R', 'X', 'T',
                        0xE0, 0, 0};
```

```
uint8 tx_resp_msg3[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '3', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8 rx_poll_msg4[] = {0x41, 0x88, 0, 0xCA, 0xDE, '4', 'R', 'X', 'T',
                        0xE0, 0, 0};
uint8 tx_resp_msg4[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'X', 'T', '4', 'R',
                        0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
/* String used to display acknowledgement phrase on screen (20 characters
 * maximum). */
char ack_str[20] = {0}; //""

uint16 pan_id = 0xDECA;
//uint8 eui[] = {'A', 'C', 'K', 'D', 'A', 'T', 'R', 'X'};
uint16 short_addr1 = 0x5231; // "R1"
uint16 short_addr2 = 0x5232; // "R2"
uint16 short_addr3 = 0x5233; // "R3"
uint16 short_addr4 = 0x5234; // "R4"

/* Default communication configuration. We use here EVK1000's mode 4. */
static dwt_config_t config = {
    2,               // Channel number.
    DWT_PRF_64M,     // Pulse repetition frequency.
    DWT_PLEN_128,    // Preamble length. Used in TX only.
    DWT_PAC8,        // Preamble acquisition chunk size. Used in RX only.
    9,               // TX preamble code. Used in TX only.
    9,               // RX preamble code. Used in RX only.
    0,               // 0 to use standard SFD, 1 to use non-standard SFD.
    DWT_BR_6M8,      // Data rate.
    DWT_PHRMODE_STD, // PHY header mode.
    (129 + 8 - 8)    // SFD timeout (preamble length + 1 + SFD length - PAC
                      size). Used in RX only.
};

/* Frame sequence number, incremented after each transmission. */
static uint8 frame_seq_nb = 0;
/* Hold copy of status register state here for reference so that it can be
 * examined at a debug breakpoint. */
static uint32 status_reg = 0;

static uint8 rx_buffer[RX_BUFFER_LEN];

/* Timestamps of frames transmission/reception.
 * As they are 40-bit wide, we need to define a 64-bit int type to handle them. */
typedef unsigned long long uint64;
static uint64 poll_rx_ts;
static uint64 resp_tx_ts;


/* Declaration of static functions. */
static uint64 get_rx_timestamp_u64(void);
static void resp_msg_set_ts(uint8 *ts_field, const uint64 ts);
static void get_distance1(void);
static void get_distance2(void);
static void get_distance3(void);
static void get_distance4(void);
static void send_wifi_ack(uint8 num);


/*! ------------------------------------------------------------------------------------
 * @fn responder()
 * @brief Setup board as a responder communicating with up to two responders using
 *        preset messages. Send acknowledgement of continued operation to UART3.
 * @param num - device number
 * @param enableWifi - input of the eighth DIP switch
 * @return none
 */
void responder(uint8 num, int enableWifi)
{
    /* Start with board specific hardware initialization. */
    peripherals_init();

#ifdef DEBUG
    /* Initialize toggle bits for debug */
    TRIS_D1 = 0;
    toggle_d1
    /* Display application name. */
    print_string("RESPONDER");
    putu('\n'); putu('\r');
#endif

    /* Reset and initialize DW1000.
     * For initialization, DW1000 clocks must be temporarily set to crystal speed.
     * After initialization SPI rate can be increased for optimum performance. */
    reset_DW1000(); // Target specific drive of RSTn line into DW1000 low for a
                      period.
    spi_set_rate_low();
    if (dwt_initialise(DWT_LOADUCODE) == DWT_ERROR)
    {

#ifdef DEBUG
    print_string("INIT FAILED");
    putu('\n'); putu('\r');
#endif

        while (1)
        { };
    }
    spi_set_rate_high();
    /* Configure DW1000. */
    dwt_configure(&config);
    /* Apply default antenna delay value. */
    dwt_setrxantennadelay(RX_ANT_DLY);
    dwt_settxantennadelay(TX_ANT_DLY);

    dwt_setpanid(pan_id);
    //dwt_seteui(eui);
    switch(num)
    {
        case 1:
            dwt_setaddress16(short_addr1);
            break;
        case 2:
            dwt_setaddress16(short_addr2);
            break;
        case 3:
            dwt_setaddress16(short_addr3);
            break;
        case 4:
            dwt_setaddress16(short_addr4);
            break;
    }
    dwt_setaddress16(short_addr);
    dwt_enableframefilter(DWT_FF_DATA_EN);
    /* Loop forever responding to ranging requests. */
    int i = 0;
    while (1)
    {
        i++;
        if(enableWifi && (i = 100))
        {
            i = 0;
            send_wifi_ack(num);
        }
        switch(num)
        {
            case 1:
                get_distance1();
                break;
            case 2:
                get_distance2();
                break;
            case 3:
                get_distance3();
                break;
            case 4:
                get_distance4();
                break;
        }
    }
}

/*! ----------------------------------------------------------------------------
 * @fn get_distance1()
 * @brief Responder 1, receive and send a message from and to the initializer
 * @param none
 * @return none
 */
static void get_distance1(void)
{
    /* Activate reception immediately. */
```

```
dwt_rxenable(DWT_START_RX_IMMEDIATE);
/* Poll for reception of a frame or error/timeout. */
while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
         SYS_STATUS_ALL_RX_ERR)))
{ };
char str[5];
if (status_reg & SYS_STATUS_RXFCG)
{
    uint32 frame_len;
    /* Clear good RX frame event in the DW1000 status register. */
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
    /* A frame has been received, read it into the local buffer. */
    frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFL_MASK_1023;
    if (frame_len <= RX_BUFFER_LEN)
    {
        dwt_readrxdata(rx_buffer, frame_len, 0);
    }
    /* Check that the frame is a poll sent by initiator.
     * As the sequence number field of the frame is not relevant, it is
     * cleared to simplify the validation of the frame. */
    rx_buffer[ALL_MSG_SN_IDX] = 0;
    if (memcmp(rx_buffer, rx_poll_msg1, ALL_MSG_COMMON_LEN) == 0)
    {
        uint32 resp_tx_time;
        int ret;
        /* Retrieve poll reception timestamp. */
        poll_rx_ts = get_rx_timestamp_u64();
        /* Compute final message transmission time. */
        resp_tx_time = (poll_rx_ts + (POLL_RX_TO_RESP_TX_DLY_UUS *
                        UUS_TO_DWT_TIME))>>8;
        dwt_setdelayedtrxtime(resp_tx_time);
        /* Response TX timestamp is the transmission time programmed plus the
         * antenna delay. */
        resp_tx_ts = (((uint64)(resp_tx_time & 0xFFFFFFFEUL))<<8) + TX_ANT_DLY;
        /* Write all timestamps in the final message. */
        resp_msg_set_ts(&tx_resp_msg1[RESP_MSG_POLL_RX_TS_IDX], poll_rx_ts);
        resp_msg_set_ts(&tx_resp_msg1[RESP_MSG_RESP_TX_TS_IDX], resp_tx_ts);
        /* Write and send the response message. */
        tx_resp_msg1[ALL_MSG_SN_IDX] = frame_seq_nb;
        dwt_writetxdata(sizeof(tx_resp_msg1), tx_resp_msg1, 0);
        /* Zero offset in TX buffer. */
        dwt_writetxfctrl(sizeof(tx_resp_msg1), 0, 1);
        /* Zero offset in TX buffer, ranging. */
        ret = dwt_starttx(DWT_START_TX_DELAYED);
        /* If dwt_starttx() returns an error, abandon this ranging exchange
         * and proceed to the next one. */
        if (ret == DWT_SUCCESS)
        {
            /* Poll DW1000 until TX frame sent event set. */
            while (!(dwt_read32bitreg(SYS_STATUS_ID) & SYS_STATUS_TXFRS))
            { };
            /* Clear TXFRS event. */
            dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
```

```
                /* Increment frame sequence number after transmission of the poll
                 * message (modulo 256). */
                frame_seq_nb++;
            }
        }
    }
    else
    {
        /* Clear RX error events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialize LDE operation. */
        dwt_rxreset();
    }
}

/*! ----------------------------------------------------------------------------
 * @fn get_distance2()
 * @brief Responder 2, receive and send a message from and to the initializer
 * @param none
 * @return none
 */
static void get_distance2(void)
{
    /* Activate reception immediately. */
    dwt_rxenable(DWT_START_RX_IMMEDIATE);
    /* Poll for reception of a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_ERR)))
    { };
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFL_MASK_1023;
        if (frame_len <= RX_BUFFER_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is a poll sent by initiator.
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_poll_msg2, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 resp_tx_time;
            int ret;
            /* Retrieve poll reception timestamp. */
            poll_rx_ts = get_rx_timestamp_u64();
            /* Compute final message transmission time. */
            resp_tx_time = (poll_rx_ts + (POLL_RX_TO_RESP_TX_DLY_UUS *
                            UUS_TO_DWT_TIME))>>8;
            dwt_setdelayedtrxtime(resp_tx_time);
            /* Response TX timestamp is the transmission time programmed plus the
             * antenna delay. */
            resp_tx_ts = (((uint64)(resp_tx_time & 0xFFFFFFFEUL))<<8) + TX_ANT_DLY;
            /* Write all timestamps in the final message. */
            resp_msg_set_ts(&tx_resp_msg2[RESP_MSG_POLL_RX_TS_IDX], poll_rx_ts);
            resp_msg_set_ts(&tx_resp_msg2[RESP_MSG_RESP_TX_TS_IDX], resp_tx_ts);
            /* Write and send the response message. */
            tx_resp_msg2[ALL_MSG_SN_IDX] = frame_seq_nb;
            dwt_writetxdata(sizeof(tx_resp_msg2), tx_resp_msg2, 0);
            /* Zero offset in TX buffer. */
            dwt_writetxfctrl(sizeof(tx_resp_msg2), 0, 1);
            /* Zero offset in TX buffer, ranging. */
            ret = dwt_starttx(DWT_START_TX_DELAYED);
            /* If dwt_starttx() returns an error, abandon this ranging exchange
             * and proceed to the next one. */
            if (ret == DWT_SUCCESS)
            {
                /* Poll DW1000 until TX frame sent event set. */
                while (!(dwt_read32bitreg(SYS_STATUS_ID) & SYS_STATUS_TXFRS))
                { };
                /* Clear TXFRS event. */
                dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
                /* Increment frame sequence number after transmission of the poll
                 * message (modulo 256). */
                frame_seq_nb++;
            }
        }
    }
    else
    {
        /* Clear RX error events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialize LDE operation. */
        dwt_rxreset();
    }
}

/*! ----------------------------------------------------------------------------
 * @fn get_distance3()
 * @brief Responder 3, receive and send a message from and to the initializer
 * @param none
 * @return none
 */
static void get_distance3(void)
{
    /* Activate reception immediately. */
    dwt_rxenable(DWT_START_RX_IMMEDIATE);
    /* Poll for reception of a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_ERR)))
    { };
    if (status_reg & SYS_STATUS_RXFCG)
```

```
{
    uint32 frame_len;
    /* Clear good RX frame event in the DW1000 status register. */
    dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
    /* A frame has been received, read it into the local buffer. */
    frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFL_MASK_1023;
    if (frame_len <= RX_BUFFER_LEN)
    {
        dwt_readrxdata(rx_buffer, frame_len, 0);
    }
    /* Check that the frame is a poll sent by initiator.
     * As the sequence number field of the frame is not relevant, it is
     * cleared to simplify the validation of the frame. */
    rx_buffer[ALL_MSG_SN_IDX] = 0;
    if (memcmp(rx_buffer, rx_poll_msg3, ALL_MSG_COMMON_LEN) == 0)
    {
        uint32 resp_tx_time;
        int ret;
        /* Retrieve poll reception timestamp. */
        poll_rx_ts = get_rx_timestamp_u64();
        /* Compute final message transmission time. */
        resp_tx_time = (poll_rx_ts + (POLL_RX_TO_RESP_TX_DLY_UUS *
                        UUS_TO_DWT_TIME))>>8;
        dwt_setdelayedtrxtime(resp_tx_time);
        /* Response TX timestamp is the transmission time programmed plus the
         * antenna delay. */
        resp_tx_ts = (((uint64)(resp_tx_time & 0xFFFFFFFEUL))<<8) + TX_ANT_DLY;
        /* Write all timestamps in the final message. */
        resp_msg_set_ts(&tx_resp_msg3[RESP_MSG_POLL_RX_TS_IDX], poll_rx_ts);
        resp_msg_set_ts(&tx_resp_msg3[RESP_MSG_RESP_TX_TS_IDX], resp_tx_ts);
        /* Write and send the response message. */
        tx_resp_msg3[ALL_MSG_SN_IDX] = frame_seq_nb;
        dwt_writetxdata(sizeof(tx_resp_msg3), tx_resp_msg3, 0);
        /* Zero offset in TX buffer. */
        dwt_writetxfctrl(sizeof(tx_resp_msg3), 0, 1);
        /* Zero offset in TX buffer, ranging. */
        ret = dwt_starttx(DWT_START_TX_DELAYED);
        /* If dwt_starttx() returns an error, abandon this ranging exchange
         * and proceed to the next one. */
        if (ret == DWT_SUCCESS)
        {
            /* Poll DW1000 until TX frame sent event set. */
            while (!(dwt_read32bitreg(SYS_STATUS_ID) & SYS_STATUS_TXFRS))
            { };
            /* Clear TXFRS event. */
            dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
            /* Increment frame sequence number after transmission of the poll
             * message (modulo 256). */
            frame_seq_nb++;
        }
    }
}
else
```

```
    {
        /* Clear RX error events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialize LDE operation. */
        dwt_rxreset();
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn get_distance4()
 * @brief Responder 4, receive and send a message from and to the initializer
 * @param none
 * @return none
 */
static void get_distance4(void)
{
    /* Activate reception immediately. */
    dwt_rxenable(DWT_START_RX_IMMEDIATE);
    /* Poll for reception of a frame or error/timeout. */
    while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) & (SYS_STATUS_RXFCG |
             SYS_STATUS_ALL_RX_ERR)))
    { };
    if (status_reg & SYS_STATUS_RXFCG)
    {
        uint32 frame_len;
        /* Clear good RX frame event in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG);
        /* A frame has been received, read it into the local buffer. */
        frame_len = dwt_read32bitreg(RX_FINFO_ID) & RX_FINFO_RXFL_MASK_1023;
        if (frame_len <= RX_BUFFER_LEN)
        {
            dwt_readrxdata(rx_buffer, frame_len, 0);
        }
        /* Check that the frame is a poll sent by initiator.
         * As the sequence number field of the frame is not relevant, it is
         * cleared to simplify the validation of the frame. */
        rx_buffer[ALL_MSG_SN_IDX] = 0;
        if (memcmp(rx_buffer, rx_poll_msg4, ALL_MSG_COMMON_LEN) == 0)
        {
            uint32 resp_tx_time;
            int ret;
            /* Retrieve poll reception timestamp. */
            poll_rx_ts = get_rx_timestamp_u64();
            /* Compute final message transmission time. */
            resp_tx_time = (poll_rx_ts + (POLL_RX_TO_RESP_TX_DLY_UUS *
                            UUS_TO_DWT_TIME))>>8;
            dwt_setdelayedtrxtime(resp_tx_time);
            /* Response TX timestamp is the transmission time programmed plus the
             * antenna delay. */
            resp_tx_ts = (((uint64)(resp_tx_time & 0xFFFFFFFEUL))<<8) + TX_ANT_DLY;
            /* Write all timestamps in the final message. */
            resp_msg_set_ts(&tx_resp_msg4[RESP_MSG_POLL_RX_TS_IDX], poll_rx_ts);
            resp_msg_set_ts(&tx_resp_msg4[RESP_MSG_RESP_TX_TS_IDX], resp_tx_ts);
            /* Write and send the response message. */
            tx_resp_msg4[ALL_MSG_SN_IDX] = frame_seq_nb;
            dwt_writetxdata(sizeof(tx_resp_msg4), tx_resp_msg4, 0);
            /* Zero offset in TX buffer. */
            dwt_writetxfctrl(sizeof(tx_resp_msg4), 0, 1);
            /* Zero offset in TX buffer, ranging. */
            ret = dwt_starttx(DWT_START_TX_DELAYED);
            /* If dwt_starttx() returns an error, abandon this ranging exchange
             * and proceed to the next one. */
            if (ret == DWT_SUCCESS)
            {
                /* Poll DW1000 until TX frame sent event set. */
                while (!(dwt_read32bitreg(SYS_STATUS_ID) & SYS_STATUS_TXFRS))
                { };
                /* Clear TXFRS event. */
                dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_TXFRS);
                /* Increment frame sequence number after transmission of the poll
                 * message (modulo 256). */
                frame_seq_nb++;
            }
        }
    }
    else
    {
        /* Clear RX error events in the DW1000 status register. */
        dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
        /* Reset RX to properly reinitialize LDE operation. */
        dwt_rxreset();
    }
}

/*! ------------------------------------------------------------------------------
 * @fn get_rx_timestamp_u64()
 * @brief Get the RX time-stamp in a 64-bit variable.
 *        /!\ This function assumes that length of time-stamps is 40 bits, for
 *            both TX and RX!
 * @param none
 * @return 64-bit value of the read time-stamp.
 */
static uint64 get_rx_timestamp_u64(void)
{
    uint8 ts_tab[5];
    uint64 ts = 0;
    int i;
    dwt_readrxtimestamp(ts_tab);
    for (i = 4; i >= 0; i--)
    {
        ts <<= 8;
        ts |= ts_tab[i];
    }
    return ts;
}
```

```c
/*! ------------------------------------------------------------------------------------
 * @fn final_msg_set_ts()
 * @brief Fill a given timestamp field in the response message with the given
 *        value. In the timestamp fields of the response message, the least
 *        significant byte is at the lower address.
 * @param ts_field - pointer on the first byte of the timestamp field to fill
 *                   ts timestamp value
 * @return none
 */
static void resp_msg_set_ts(uint8 *ts_field, const uint64 ts)
{
    int i;
    for (i = 0; i < RESP_MSG_TS_LEN; i++)
    {
        ts_field[i] = (ts>>(i * 8)) & 0xFF;
    }
}

/*! ------------------------------------------------------------------------------------
 * @fn send_wifi_ack()
 * @brief Responder sends an acknowledgement phrase through UART3
 * @param num - device number
 * @param display message on MQTT server
 * @return none
 */
static void send_wifi_ack(uint8 num)
{
    sprintf(ack_str, "RA<%d: STAYING ALIVE>", num);
    print_string3(ack_str);
}
```

3. Main Function

```
/*
 * @file main.c
 * @author Katherine Sanders
 *         Eddie Hunckler
 *         Stephen McAndrew
 * @brief Program to run RFPS system
 *
 * edited 26.4.17
 */

#include "deca_device_api.h"
#include "deca_regs.h"
#include "functions17.h"
#include "lcd.h"
#include "magnetometer.h"
#include "port.h"
#include "SDlib16.h"
#include "sleep.h"

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned long long uint64;
uint8 switch1;
uint8 switch2;
uint8 switch3;
int enableWifi;
uint8 num;

void main(void)
{
    serial_init(115200);
    serial_3_init(115200u);
    MAG3110_init();

    /* Get the data from the configuration switches */
    configDIP();
    enableWifi = DIP8;
    switch3 = DIP3;
    switch2 = DIP2;
    switch1 = DIP1;

    /* Use switch positions to determine module type - output to UART1 and UART3 */
    determine_parameters(switch1, switch2, switch3);
    MEASUREMENT_PIN = 1;

    if(switch1 == 0)
    {
        initializer(enableWifi); //output to UART1, UART3
    }
    else
    {
        responder(num, enableWifi); //output to UART3
    }
}
```