SOURCE CODE GUIDE

DECARANGERTLS ARM SOURCE CODE

Understanding and using the DecaRangeRTLS ARM source code

Version 2.3

This document is subject to change without notice.

DOCUMENT INFORMATION

Disclaimer

Decawave reserves the right to change product specifications without notice. As far as possible changes to functionality and specifications will be issued in product specific errata sheets or in new versions of this document. Customers are advised to check the Decawave website for the most recent updates on this product

LIFE SUPPORT POLICY

Decawave products are not authorized for use in safety-critical applications (such as life support) where a failure of the Decawave product would reasonably be expected to cause severe personal injury or death. Decawave customers using or selling Decawave products in such a manner do so entirely at their own risk and agree to fully indemnify Decawave and its representatives against any damages arising out of the use of Decawave products in such safety-critical applications.

Caution! ESD sensitive device.
Precaution should be used when handling the device in order to prevent permanent damage

DISCLAIMER

This Disclaimer applies to the DecaRanging RTLS-ARM source code and the DecaRanging RTLS-PC source code (collectively “Decawave Software”) provided by Decawave Ltd. (“Decawave”).

Downloading, accepting delivery of or using the Decawave Software indicates your agreement to the terms of this Disclaimer. If you do not agree with the terms of this Disclaimer do not download, accept delivery of or use the Decawave Software.

Decawave Software incorporates STSW-STM32046 (STM32F105/7, STM32F2 and STM32F4 USB on-the-go Host and Device library (UM1021)) software (“STM Software”) provided to Decawave by ST Microelectronics (“STM”) under STM’s Liberty V2 software license agreement dated November 16th 2011 available here (“STM Software License Agreement”). Downloading, accepting delivery of or using STM Software as incorporated in Decawave Software indicates your agreement to the terms of the STM Software License Agreement and in particular the requirement that the STM Software be used only with STM microcontrollers and not with microcontrollers from any other manufacturer. If you do not wish to accept the terms of the STM Software License Agreement then you may still use the Decawave Software on the condition that you do not use the STM Software incorporated therein.

Decawave Software is solely intended to assist you in developing systems that incorporate Decawave semiconductor products. You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your systems and products. THE DECISION TO USE DECAWAVE SOFTWARE IN WHOLE OR IN PART IN YOUR SYSTEMS AND PRODUCTS RESTS ENTIRELY WITH YOU.

DECAWAVE SOFTWARE IS PROVIDED "AS IS". DECAWAVE MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE DECAWAVE SOFTWARE OR USE OF THE DECAWAVE SOFTWARE, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. DECAWAVE DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO DECAWAVE SOFTWARE OR THE USE THEREOF.

DECAWAVE SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON THE DECAWAVE SOFTWARE OR THE USE OF THE DECAWAVE SOFTWARE WITH DECAWAVE SEMICONDUCTOR TECHNOLOGY. IN NO EVENT SHALL DECAWAVE BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, INCLUDING WITHOUT LIMITATION TO THE GENERALITY OF THE FOREGOING, LOSS OF ANTICIPATED PROFITS, GOODWILL, REPUTATION, BUSINESS RECEIPTS OR CONTRACTS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION), LOSSES OR EXPENSES RESULTING FROM THIRD PARTY CLAIMS. THESE LIMITATIONS WILL APPLY REGARDLESS OF THE FORM OF ACTION, WHETHER UNDER STATUTE, IN CONTRACT OR TORT INCLUDING NEGLIGENCE OR ANY OTHER FORM OF ACTION AND WHETHER OR NOT DECAWAVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE.

You are authorized to use Decawave Software in your end products and to modify the Decawave Software in the development of your end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER DECAWAVE INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Decawave semiconductor products or Decawave Software are used.

You acknowledge and agree that you are solely responsible for compliance with all legal, regulatory and safety-related requirements concerning your products, and any use of Decawave Software in your applications, notwithstanding any applications-related information or support that may be provided by Decawave.

Decawave reserves the right to make corrections, enhancements, improvements and other changes to its software at any time.

Mailing address: -

Decawave Ltd.,
Adelaide Chambers,
Peter Street,
D08 T6YA,
Dublin 8

TABLE OF CONTENTS

1 OVERVIEW

This document, "*DecaRangeRTLS ARM Source Code Guide*" is a guide to the application source code of Decawave's "TREK1000" two-way ranging RTLS demonstration application running on the ARM microcontroller on the EVB1000 development platform.

This document should be read in conjunction with the "*TREK1000 User Manual*" which gives an overview of DW1000 RTLS evaluation kit (TREK1000) and describes how to operate the DecaRangeRTLS Application.

This document discusses the source code of the DecaRangeRTLS ARM application, covering the structure of the software and the operation of the ranging RTLS demo application particularly the way the range is calculated.

Section 6 is written in the style of a walkthrough of execution flow of the software. It should give a good understanding of the basic operational steps of transmission and reception, which in turn should help integrating/porting the ranging function to customers platforms.

This document relates to the following versions: `"DecaRangeRTLS ARM 2.25"` application version and `"DW1000 Device Driver Version 04.00.05"` driver version. The device driver version information may be found in source code file "deca_version.h", and the application version is specified in "dw_main.c".

Figure 1 below shows the layered structure of the DecaRangeRTLS application, giving the names of the main files associated with each layer and a brief description of the functionality provided at that layer.

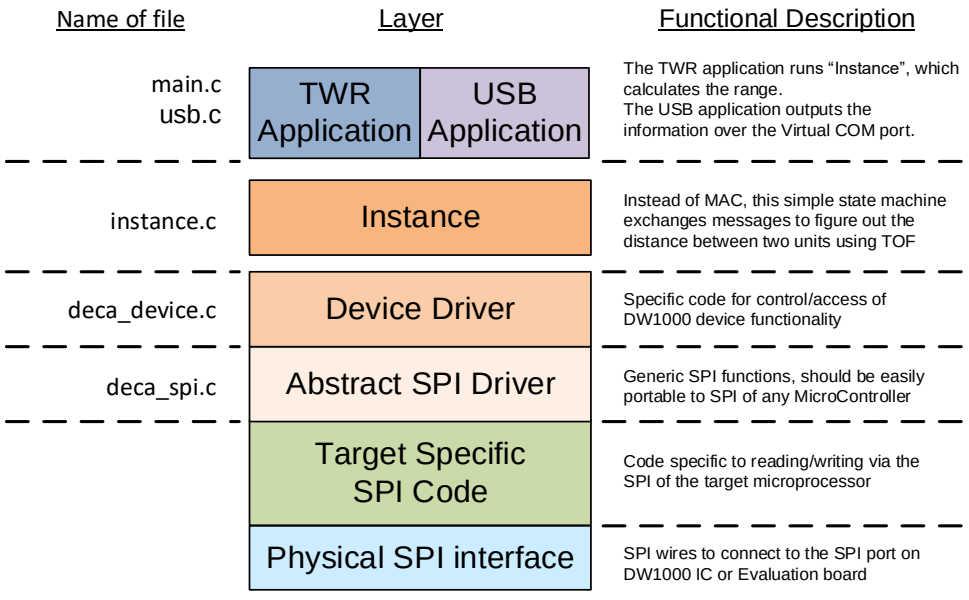

Figure 1: Software layers in DecaRange RTLS ARM application

The layers, functions and files involved are described in the following section.

2 DESCRIPTION OF DECARANGERTLS ARM CODE STRUCTURE

With reference to Figure 1, the identified layers are described in more detail below.

2.1 Target Specific Code

In the Cube MX version of the project, the ST Microelectronics Cube MX tool is used to generate the low-level ARM (HAL) code. The *main.c* contains the various peripheral initialisations and definitions.

In the CooCox IDE based project the low-level ARM specific code can be found in \src\platform\ – the two files *port.c* and *port.h* define target peripherals and GPIOs which are enabled and in use i.e. SPI1 for SPI communications with DW1000, SPI2 for SPI communications with LCD, other GPIO lines for application configuration and control.

SPI1:

```
#define SPIx                        SPI1
#define SPIx_GPIO                   GPIOA
#define SPIx_CS                     GPIO_Pin_4
#define SPIx_CS_GPIO                GPIOA
#define SPIx_SCK                    GPIO_Pin_5
#define SPIx_MISO                   GPIO_Pin_6
#define SPIx_MOSI                   GPIO_Pin_7
```

The SPI1 peripheral is used to communicate to the DW1000 SPI bus.

Interrupt line:

```
#define DECAIRQ                     GPIO_Pin_8
#define DECAIRQ_GPIO                GPIOA
#define DECAIRQ_EXTI                EXTI_Line8
#define DECAIRQ_EXTI_PORT           GPIO_PortSourceGPIOA
#define DECAIRQ_EXTI_PIN            GPIO_PinSource8
#define DECAIRQ_EXTI_IRQn           EXTI9_5_IRQn
#define DECAIRQ_EXTI_USEIRQ         ENABLE
```

The DW1000 interrupt line is connected to GPIOA pin 8. Note: For MP the line is active high.

LCD driver:

```
#define SPIy                        SPI2
#define SPIy_GPIO                   GPIOB
#define SPIy_CS                     GPIO_Pin_12
#define SPIy_CS_GPIO                GPIOB
#define SPIy_SCK                    GPIO_Pin_13
#define SPIy_MISO                   GPIO_Pin_14
#define SPIy_MOSI                   GPIO_Pin_15
```

The SPI2 peripheral is used to communicate with the LCD.

Application configuration switches (S1):

```
#define TA_SW1_3                    GPIO_Pin_0
#define TA_SW1_4                    GPIO_Pin_1
#define TA_SW1_5                    GPIO_Pin_2
#define TA_SW1_6                    GPIO_Pin_3
#define TA_SW1_7                    GPIO_Pin_4
#define TA_SW1_8                    GPIO_Pin_5
#define TA_SW1_GPIO                 GPIOC
```

Configuration switch (**S1**) is used to choose between the *Anchor* and *Tag* modes and various channel configurations. See "*TREK1000 User Manual*" for more details.

The src\compiler\compiler.h contains the standard library files which can be replaced if desired, (e.g. if one wishes to use functions optimised for smaller code size, say).

2.2 Abstract SPI Driver – SPI Level code

The file *src\platform\deca_spi.c* provides abstract SPI driver functions *openspi()*, *closespi()*, *writetospi()* and *readfromspi()*. These are mapped onto the ARM microcontroller SPI interface driver.

2.3 Device Driver – DW1000 Device Level Code

The file deca_device_api.h provides the interface to a library of API functions to control and configure the DW1000 registers and implement functions for device level control. The API functions are described in the "*DW1000 Device Driver Application Programming Interface (API) Guide*" document.

2.4 Instance Code

The instance code (in *src\application*\instance_*.c) provides a two-way ranging RTLS demonstration application. This instance code sits where the MAC and application would normally reside. For expediency in developing the ranging RTLS demonstration to showcase ranging and performance of the DW1000, the ranging RTLS demo application was implemented directly on top of the DW1000 driver API.

The instance runs in different modes (*Tag* or *Anchor*) depending on the role configuration set at the application layer. The *Tag* and *Anchor* modes operate as a pair to provide the two-way ranging demo functionality between two units. The two-way ranging RTLS demo application is implemented by the companion state machines in functions *tag_app_run() and anch_app_run()*, called from function *tag_run() and anch_run()*, which is the main entry point for running the instance code. In TREK single tag ranges up to 4 anchors and then a separate DecaRangeRTLS PC application can then use the ranges to calculate tag's location relative to anchors' and display on the GUI.

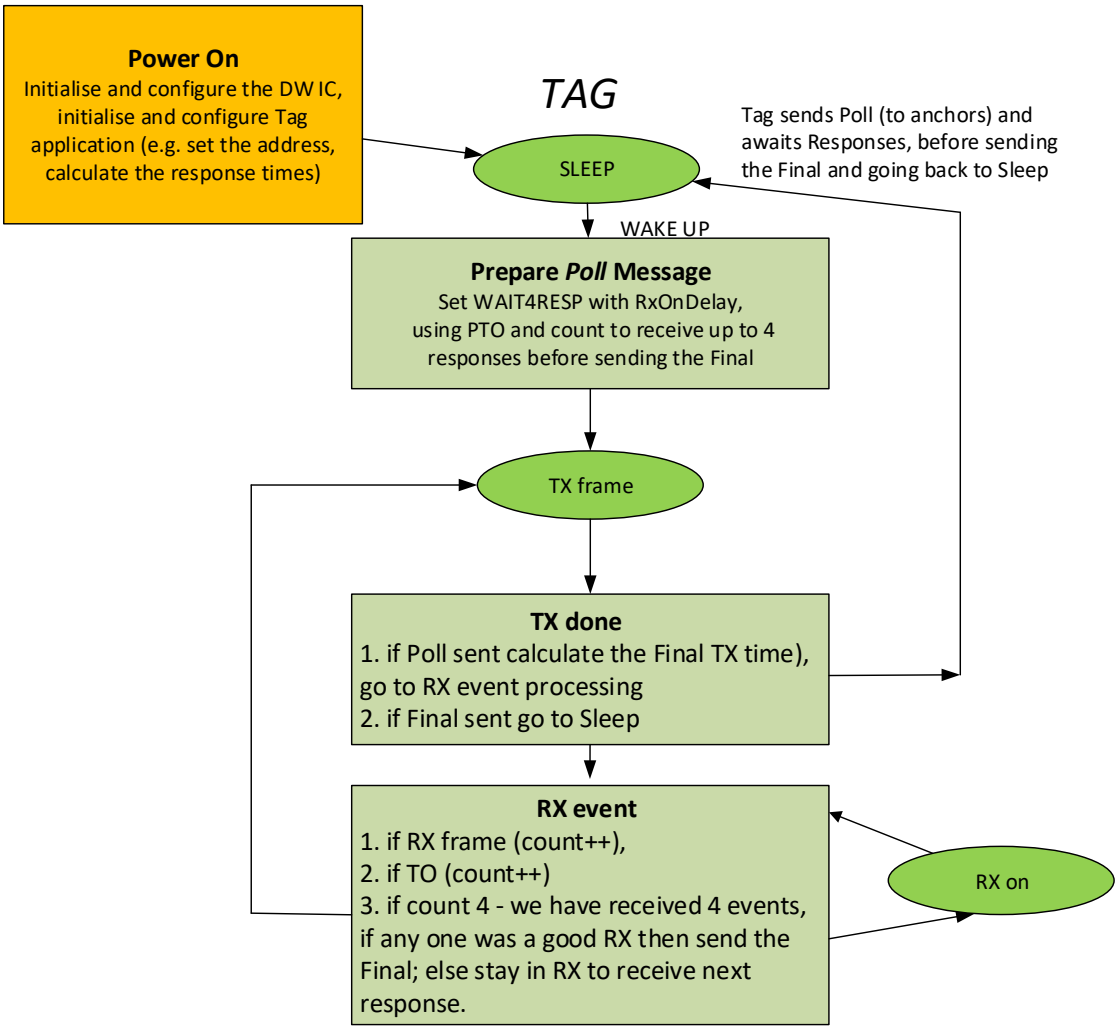

Figure 2: Brief illustration of tag instance operation

2.4.1 Operating mode – Tag

Once powered on the tag unit will try and range to 4 anchors and then go to sleep. After a period (superframe/scheduling period, T_{sf}) it will wake up and range to 4 anchors again. The tag ranges to all 4 anchors simultaneously (as described in Figure 3 below). It sends a Poll as a broadcast message (destination address is set to 0xFFFF), receives any responses, and then sends a Final, again as a broadcast. This is outlined in Figure 2.

The default scheduling period is 280 ms for 110 kbps or 100 ms for 6.81 Mbps mode. This is the period from the time that the tag sends its poll, completes the ranging exchanges or times out to the same instant of sending the poll again. These scheduling period timings are related to the duration of the ranging interactions and the number of interaction slots defined for the superframe, (described below). Note that the 6.81 Mbps mode could have a faster update rate, but a 10 Hz update rate was decided and defined for the 6.81 Mbps mode superframe.

Figure 3 gives an overview of the superframe structure and the frame timings are given in Figure 8 and Figure 9.

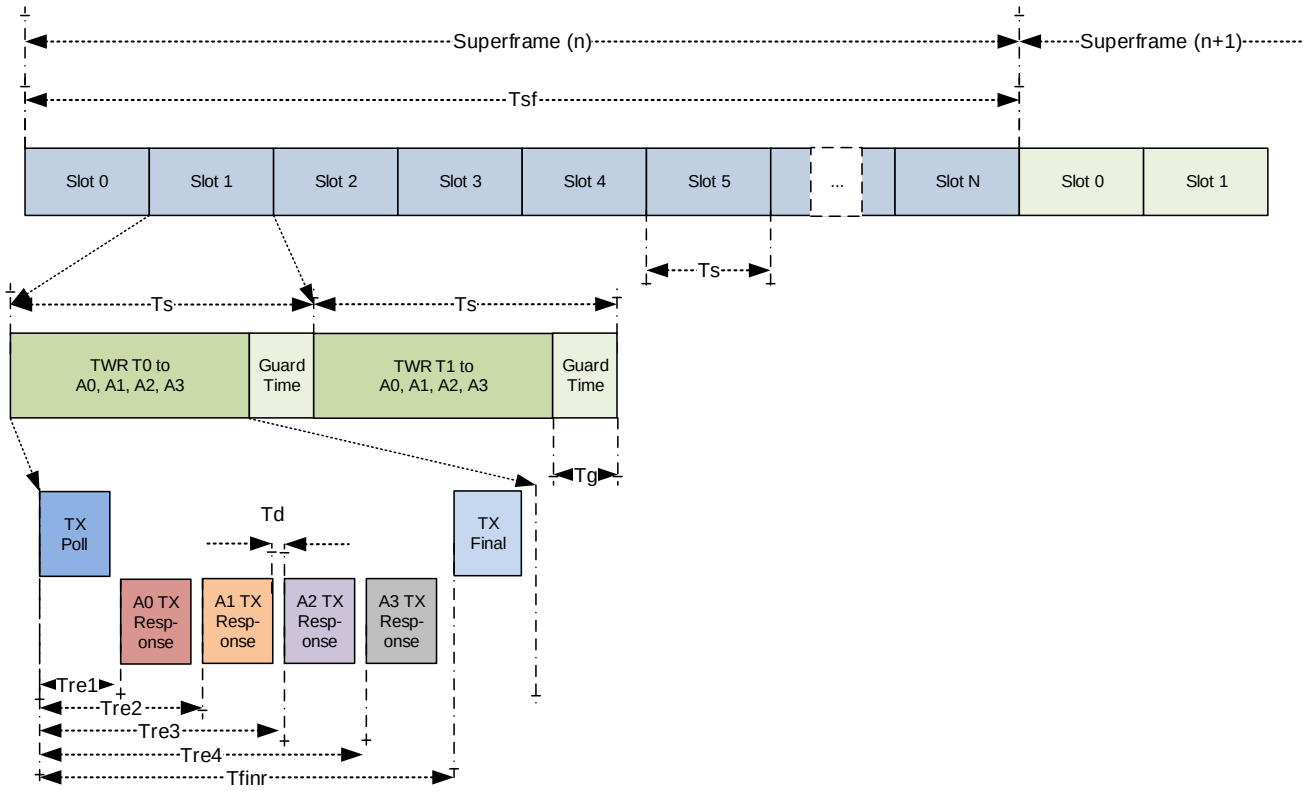

Figure 3: Tag TWR RTLS time profile

To support multiple tags (without interfering), we are using a TDMA approach. In our TREK example modes there are 10 slots in the superframe. Each slot is 28 ms in 110 kbps modes and 10 ms in 6.81 Mbps modes, and, as we assign one slot to each tag a maximum of 8 tags can be supported. Two slots are reserved for anchor to anchor ranging.

The anchor #0, is responsible for assigning and maintaining tags into their own slots.

2.4.2 Operating mode – Anchor #0

Firstly, Anchor #0 operates to assign activity slots to each of the tags so that their ranging exchanges do not mutually interfere. Anchor #0 does this by including a sleep time adjustment value every time it responds to a tag's ranging attempt. This sleep time adjustment is calculated to position each of the tags to wake-up in a separate slot, based on their address (#0 to #7). This is outlined in Figure 4Figure 2.

Secondly, Anchor #0 reports the results of all ranging exchanges via its USB port. It does this by listening for and receiving time-of-flight results that the other anchors embed in their response messages, and gathering these along with its own calculated TOF results, before sending a full set of ranging results for each tag to and attached PC via its USB port.

Anchor #0 also starts anchor to anchor ranging, which is used for auto-positioning feature. Last two slots of the superframe are used for this. Anchor #0 initially ranges to anchors #1 and #2 and then anchor #1 ranges

to anchor #2. The resultant six ranges are outputted over the USB so that the anchor positions relative to each other can be calculated.

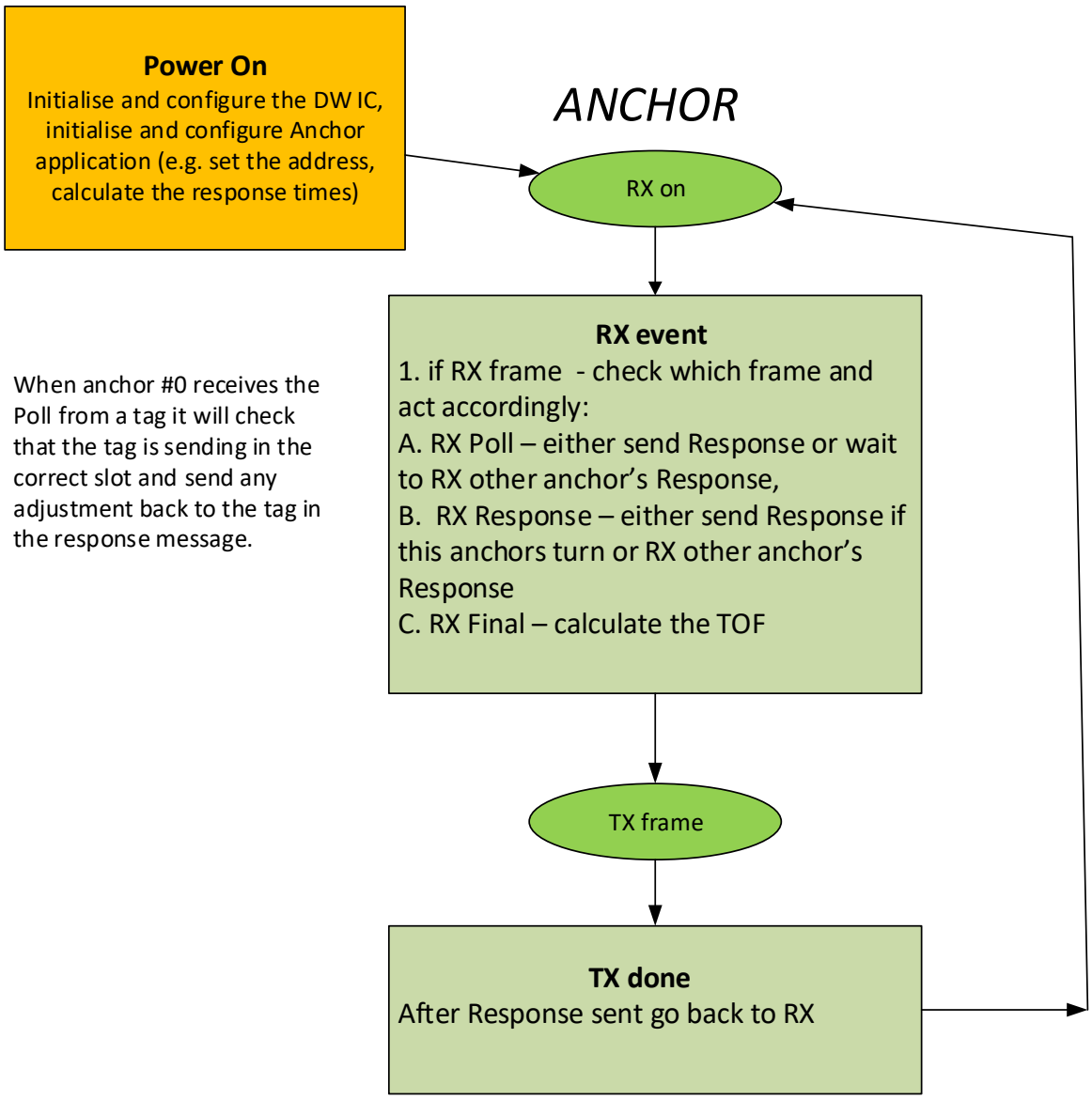

Figure 4: Brief illustration of anchor instance operation

2.4.3 Operating mode – Anchors #1 and #2

Anchor #1 and #2 are involved in tags to anchor ranging and also anchor to anchor ranging.

2.4.4 Operating mode – Anchor #3

Anchor #3 is only involved in the tag to anchor ranging. It ignores any anchor to anchor ranging messages.

2.4.5 Range Result

The ranging results are output via the USB port. Tags will report any ranges results it receives from the anchors it ranges with, as returned to it by the anchor's Response message.

All the anchor will report ranges they calculate for the tags that range with them, but also any range reports they receive from other anchors (all will receive TOFs inside the response messages). No frame filtering is used so any messages on the air can be received.

2.4.6 Ranging Method

The ranging method uses a set of three messages to complete two-round trip measurements from which the range is calculated. As messages are sent and received the *DecaRangeRTLS* ARM application retrieves the message send and receive times from the DW1000. These transmit and receive timestamps are used to work out a round trip delay and calculate the range. Figure 5 shows the arrangement and general operation of the two-way ranging as implemented by the DecaRangeRTLS ARM application.

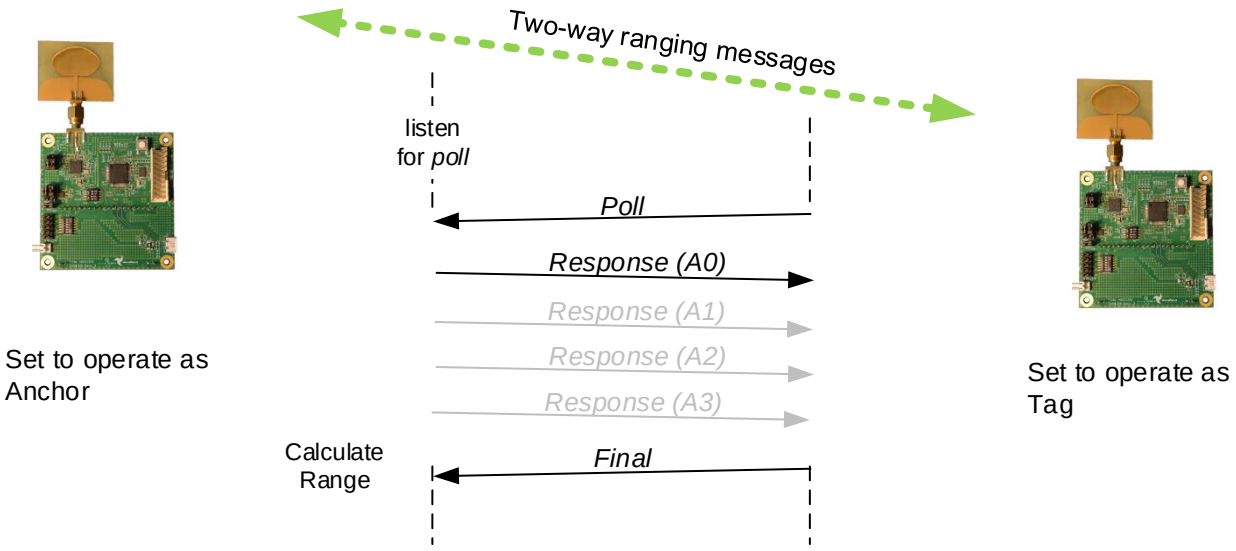

Figure 5: Two way ranging in DecaRangeRTLS ARM

Above this instance level is the application that provides the user interface described in section 2.5 below. In addition to the *tag_run() and anch_run()* function, the instance code provides control functions to the application. These functions are listed here to give the reader a quick idea of the functionality: -

instance_init(), instance_config(), instance_get_role(), instance_set_antennadelays(), instance_set_replydelay(), rx_ok_cb_tag(), rx_ok_cb_anch(), tag_process_rx_timeout() etc.

The reader is directed to the code in file instance_*.c for more details of these functions.

2.5 Top level Application code

The top level application (dw_main.c) contains the main entry point for the DecaRanging ranging demo application and also all the user interface code.

The ability of the application layer to display results depends on the capability of the hardware platform. On the EVB1000 evaluation board the LCD is used to display the resultant range from a range measurement.

Table 1: LCD display messages in the DecaRange RTLS application

Filename	Brief description
1234567890123456	This row is just for sizing the fixed space font so that the text defined here fits the 16x2 character display on the EVB1000
USB-to-SPI	Unit is in USB to SPI conversion mode
DecaRangeRTLS L2 T3 xxxxxxxxxxxx	Unit is in RTLS, "L" (Long Range) mode on channel "2", and operating as a Tag #3, see below for definition of xxxxxxxxxxxx.
DecaRangeRTLS S5 A0 xxxxxxxxxxxx	Unit is in RTLS, "S" (Short Frame) mode on channel "5", and operating as Anchor #0, see below for definition of xxxxxxxxxxxx.
DecaRangeRTLS L2 LS xxxxxxxxxxxx	Unit is in RTLS, "L" (Long Range) mode on channel "2", and operating as a Listener, see below for definition of xxxxxxxxxxxx.
AiTj:rrr.rrm	Where xxxxxxxxxxxx is the text (left) that shows information extracted from the last ranging report received by the unit, where: i - is the anchor address (least significant nibble), j - is the tag address (least significant nibble), rrr.rr - is the range between this tag and anchor in meters to two decimal places, and, the characters "A", "T", ":" and "m" are just these characters.
Continuous TX L2 Spectrum Test	Indicates that continuous transmission test mode is active. In this case operating with LR (Long Range) frame format on channel 2.

2.6 Format of ranging results as sent to USB/Virtual COM port

The application also outputs ranging and some debug information over the virtual COM port. Figure 6 shows example output from anchor 0 as viewed on Teraterm terminal emulator (communication program).

The windows PC driver is available from the ST Microelectronics website, see TREK1000 User Manual for details on its installation.

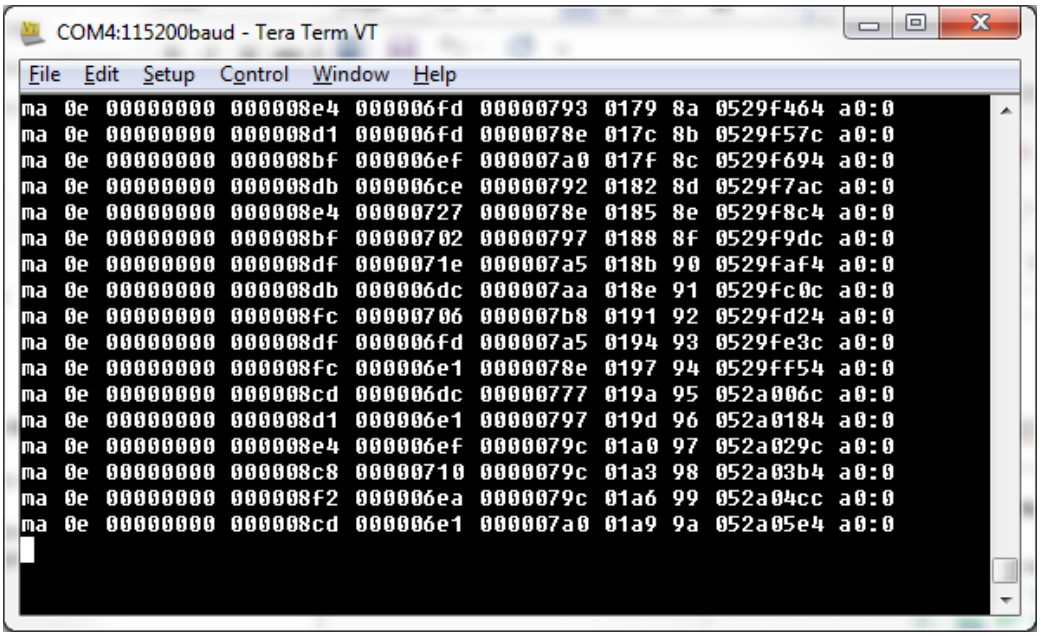

Figure 6: Example Teraterm window showing the debug info sent via COM port

There are three ranging report messages sent over the USB port:

MID MASK RANGE0 RANGE1 RANGE2 RANGE3 NRANGES RSEQ DEBUG aT:A

```
1. mr 0f 000005a4 000004c8 00000436 000003f9 0958 c0 40424042 a0:0
2. ma 07 00000000 0000085c 00000659 000006b7 095b 26 00024bed a0:0
3. mc 0f 00000663 000005a3 00000512 000004cb 095f c1 00024c24 a0:0
```

The “mr” message consists of tag to anchor raw ranges, “mc” tag to anchor range bias corrected ranges – used for tag location and “ma” anchor to anchor range bias corrected ranges – used for anchor auto-positioning.

MID	this is the message ID, as described above: mr, mc and ma
MASK	this states which RANGEs are valid, if MASK=7 then only RANGE0, RANGE1 and RANGE2 are valid (in hex, 8-bit number)
RANGE0	this is tag to anchor ID 0 range if MID = mc/mr (in mm, 32-bit hex number)
RANGE1	this is tag to anchor ID 1 range if MID = mc/mr or anchor 0 to anchor 1 range if MID = ma (in mm, 32-bit hex number)
RANGE2	this is tag to anchor ID 2 range if MID = mc/mr or anchor 0 to anchor 2 range if MID = ma (in mm, 32-bit hex number)
RANGE3	this is tag to anchor ID 3 range if MID = mc/mr or anchor 1 to anchor 2 range if MID = ma (in mm, 32-bit hex number)
NRANGES	this is a number of ranges completed by reporting unit raw range (16-bit hex number)
RSEQ	this is the range sequence number (8-bit hex number)
DEBUG	this is the TX/RX antenna delays (if MID = ma) – two 16-bit numbers or time of last range reported – if MID = mc/mr (32 bit hex number)
aT:A	the T is the tag ID and A id the anchor ID

2.7 Complete list of source code files

Table 2 gives a list of the main files that make up the source code of the DecaRangeRTLS ARM application. The file name is given along with a brief description of the file and its purpose. The reader is referred to the other sections of this document for more details on the code structure and organisation.

Table 2: List of source files in the DecaRangeRTLS ARM application

Filename	Brief description
deca_version.h	Decawave’s version number for the DW1000 driver/API code
deca_device.c	Device level Functions – source code
deca_device_api.h	Device level Functions – header
deca_mutex.c	Place holder for IRQ disable for mutual exclusion – source code
deca_param_types.h	Header defining the parameter and configuration structures
deca_params_init.c	Initialisation of configuration data for setting up the DW1000
deca_range_tables.c	Contains the ranging correction tables
deca_regs.h	Device level – header (Device Register Definitions)
deca_spi.c	SPI interface driver – source code
deca_spi.h	SPI interface driver – header
dma_spi.c	SPI interface driver – source code for DMA implementation
deca_types.h	Data type definitions – header

Filename	Brief description
port.c	ARM peripheral and GPIO configuration
port.h	ARM peripheral and GPIO configuration definitions
dw_main.c	Application – source code (main line)
main.c	Application – source code (main line)
Instance_calib.c	Calibration functions and data for the application – source code
Instance_common.c	Common application functions – source code
Instance_tag.c	Ranging Application Instance – source code
Instance_anch.c	Ranging Application Instance – source code
instance.h	Ranging Application Instance – header
compiler.h	Contains the standard library files
stm32f10x_conf.h	STM library configuration/inclusion files
stm32f10x_it.c	Interrupt handlers are defined here.
stm32f10x_it.h	Interrupt handlers are declared here.
deca_usb.c	The USB application – input/output over Virtual COM port

3 RANGING ALGORITHM

This section describes the ranging algorithm used in the DecaRangeRTLS demo application. Tag will attempt to range to four anchors and then go into DEEP SLEEP mode. (It will be woken up after superframe period to start the cycle again.)

3.1 DecaRangeRTLS ARM application's Tag/Anchor Two-way ranging algorithm

For this algorithm one end acts as a tag, periodically initiating a range measurement, while the other end acts as an anchor listening and responding to the tag and calculating the range.

In the ranging scheme the tag sends a Poll message which is received by three (or four) anchors in the infrastructure. The anchors reply in successive responses with packets RespA, RespB and RespC after which the tag sends the Final message received by all the anchors. This allows the tag to be located after sending only 2 messages and receiving 3. This scheme is illustrated below.

This represents a substantial saving in message traffic thereby saving battery power and air-time. In the DecaRangeRTLS demo the anchor sends a ranging report of the calculated range to the tag so that it knows the range too, this is done inside the next Response message. This means that a location engine can be used on the tag's side to work out tags position relative to anchors (Navigational mode or Geo-Fencing mode).

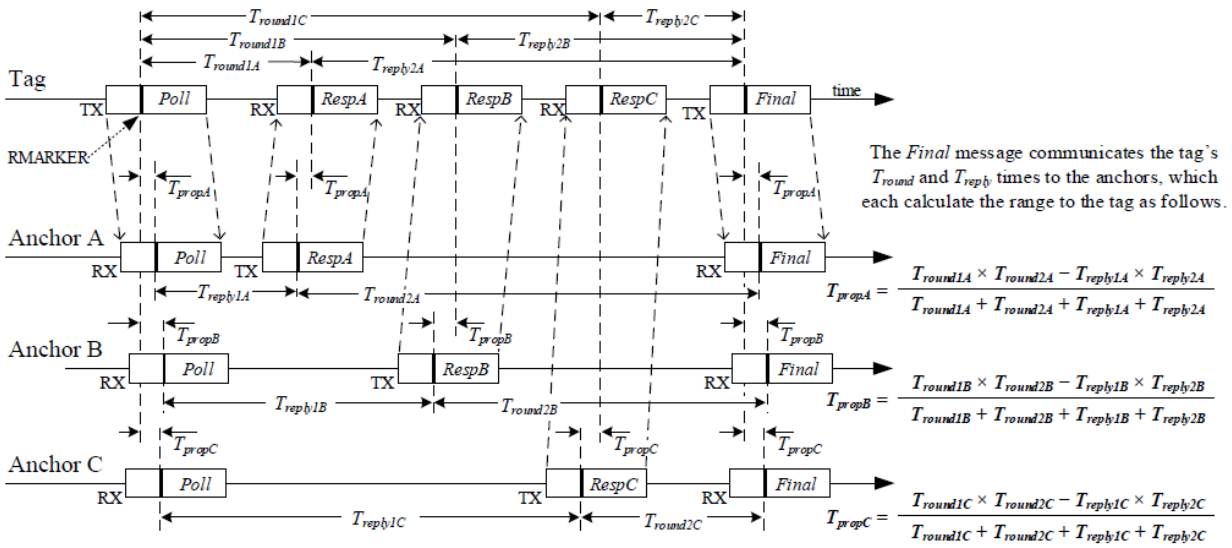

Figure 7: Range calculation in DecaRangeRTLS

3.2 Practical considerations for ranging in a real product

The application note APS016: "*Moving from TREK to a (commercial TWR RTLS) product*" should be read by anyone who would like to develop a commercial TWR RTLS product. The document aims to give an appreciation and overview of the steps involved in developing a commercial RTLS product starting from Decawave's TREK1000 Two-Way-Ranging (TWR) RTLS IC Evaluation Kit, which uses Decawave's DW1000 ultra-wideband (UWB) transceiver IC.

3.3 Frame Time Adjustments

Successful ranging relies on the system being able to accurately determine the TX and RX times of the messages as they leave one antenna and arrive at the other antenna. This is needed for antenna-to-antenna time-of-flight measurements and the resulting antenna-to-antenna distance estimation.

The significant event making the TX and RX times is defined in IEEE 802.15.4 as the "Ranging Marker (RMARKER): The first ultra-wide band (UWB) pulse of the first bit of the physical layer (PHY) header (PHR) of a ranging frame (RFRAME)". The time stamps should reflect the time instant at which the RMARKER leaves or arrives at the antenna. However, it is the digital hardware that marks the generation or reception of the RMARKER, so adjustments are needed to add the TX antenna delay to the TX timestamp, and, subtract the RX antenna delay from the RX time stamp.

The EVB1000 units as part of the TREK1000 kit have the antenna delays calibrated, and programmed into the DW1000 OTP (one-time-programmable) memory. However, if DecaRangeRTLS SW is downloaded on an EVB1000 which has not been calibrated for TREK, the application will use the default antenna delay value as set in the instance_calib.c file, there are two values, one for each channel option (2/5). The value specified is divided equally between TX and RX antenna delays. The default value has been calculated by averaging the calibration values from a number of EVB1000 boards.

4 MESSAGES USED IN TREK1000 TWR

As shown in Figure 5, three messages are employed in the two-way ranging: the Poll message, the Response message, and the Final message. The detailed formats of the messages are documented below. These follow IEEE message encoding conventions, but these are NOT standardised RTLS messages. The reader is referred to the ISO/IEC 24730-62 international standard for details of standardised message formats for use in RTLS systems based on IEEE 802.15.4 UWB.

4.1 General ranging frame format

The general message format is the IEEE 802.15.4 standard encoding for a data frame. Figure 8 shows this format. The two byte Frame Control octets are constant for the TREK application because it always uses data frames with 2-octet (16-bit) source and destination addresses, and a single 16-bit PAN ID (value 0xDECA). In a real 802.15.4 network, the PAN ID might be negotiated as part of associating with a network or it might be a defined constant based on the application.

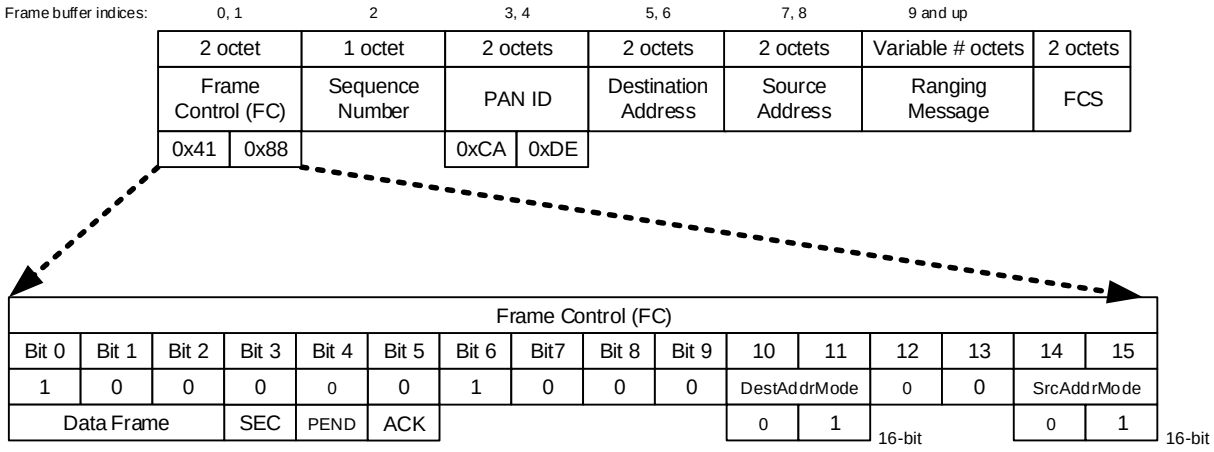

Figure 8: General ranging frame format

The sequence number octet is incremented modulo-256 for every frame sent.

The source and destination addresses are 16-bit values based on the EVB1000 board's configuration switches settings selecting the mode as tag or anchor and the tag/anchor number.

The 2-octet FCS is a CRC frame check sequence. This is generated automatically by the DW1000 IC (under software control) and appended to the transmitted message.

The content of the ranging message portion of the frame depends on which of the four ranging messages it is. These are shown in Figure 9 and described in sections 4.2 to 4.5. In these only the Ranging Message portion of the frame is shown and discussed. This data is encapsulated in the general ranging frame format of Figure 8 to form the complete ranging message in each case.

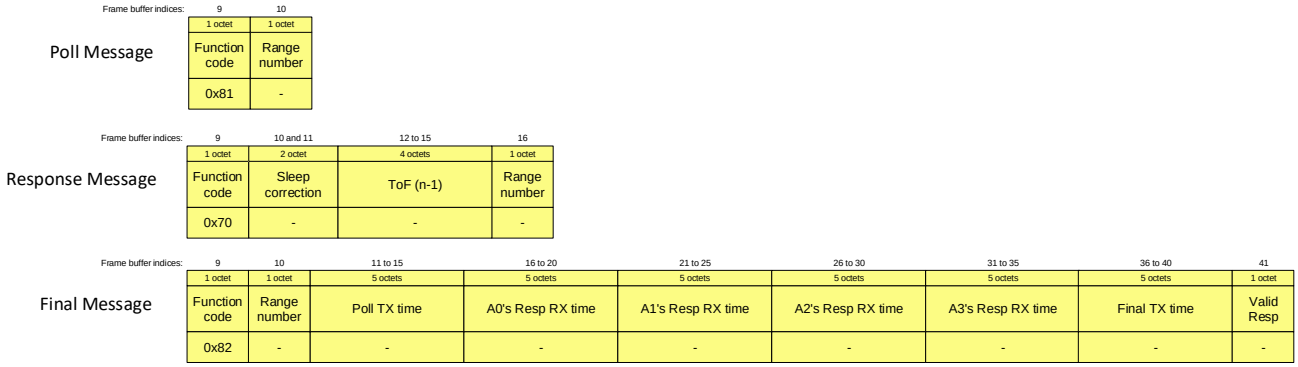

Figure 9: Ranging message encodings

4.2 Poll message

The Poll message is sent by the tag to initiate a range measurement. Table 3 lists and describes the individual fields within the Poll message.

Table 3: Fields within the ranging Poll message

Octet #'s	Value	Description
1	0x81	Function code: This octet 0x81 identifies this as a tag Poll message
2	-	Range number: This is a range sequence number, on each wake up this number is incremented.

4.3 Response message

The response message is sent by the anchor in Response to a poll from the tag. Table 4 lists and describes the individual fields within the Response message.

Table 4: Fields within the ranging Response message

Octet #'s	Value	Description
1	0x70	Function code: This octet 0x70 identifies this as the Response message
2, 3	-	Sleep correction: This two octet parameter is a correction factor that adjusts the Tag's sleep duration so that the Tag's ranging activity can be assigned and aligned into a slot that does not interfere with other tags in the system. Anchor #0, the gateway anchor, will control/set this field. All other anchors set this field to 0.
12 to 15	-	32-bit TOF from the previous exchange, corresponding to the range number as given in the next octet
16	-	Range number: This is a range sequence number, corresponding to the reported TOF.

4.4 Final message

The Final message is sent by the tag after receiving the anchor's Response message. The Final message is 44 octets in length. Table 5 lists and describes the individual fields within the Final message.

Table 5: Fields within the ranging Final message

Octet #'s	Value	Description
1	0x82	Function code: This octet identifies the message as the tag Final message
2	-	Range number: This is a range sequence number, corresponding to the range number as sent in the Poll.
3 to 7	-	Poll TX time: This 5 octet field is the TX timestamp for the tag's poll message, i.e. the precise time the frame was transmitted.
8 to 12	-	Resp RX time: This 5 octet field is the RX timestamp for the response from anchor 0, i.e. the time the tag received the response frame from the anchor.
13 to 17	-	Resp RX time: This 5 octet field is the RX timestamp for the response from anchor 1, i.e. the time the tag received the response frame from the anchor.
18 to 22	-	Resp RX time: This 5 octet field is the RX timestamp for the response from anchor 2, i.e. the time the tag received the response frame from the anchor.
23 to 27	-	Resp RX time: This 5 octet field is the RX timestamp for the response from anchor 3, i.e. the time the tag received the response frame from the anchor.
28 to 32	-	Final TX time: This 5 octet field is the TX timestamp of this final message, i.e. the time the frame was transmitted, (this is pre-calculated by the tag).
33	-	8-bit value specifying which response times are valid. Anchors that receive the final should only calculate the TOF if the response time is shown as being valid.

As well as sending the TOF to the tag, each anchor also reports the ranging result via its USB port.

4.5 Message timings

TREK demo supports following modes:

a) 110 kbps data rate with 1024 preamble length and 16 MHz PRF, using non-standard SFD of 64 symbols
b) 6.81 Mbps data rate with 128 preamble length and 16 MHz PRF and using standard SFD of 8 symbols.

The message lengths (in bytes) as shown in sections above are: Poll = 13, Response = 19, and Final = 44. If we take the longest message, the total frame duration is 4.929 ms for the 110 kbps mode and 0.214 ms for the 6.81 Mbps mode.

Table 6: Slot timings

Data Rate	N Slots	T_{re1} (µs)	T_{re2} (µs)	T_{re3} (µs)	T_{re4} (µs)	T_{finr} (µs)	T_s min (ms)
110 kbps	8	2620	5720	8820	11920	16000	26
6.81 Mbps	8	320	658	995	1335	1800	2.25

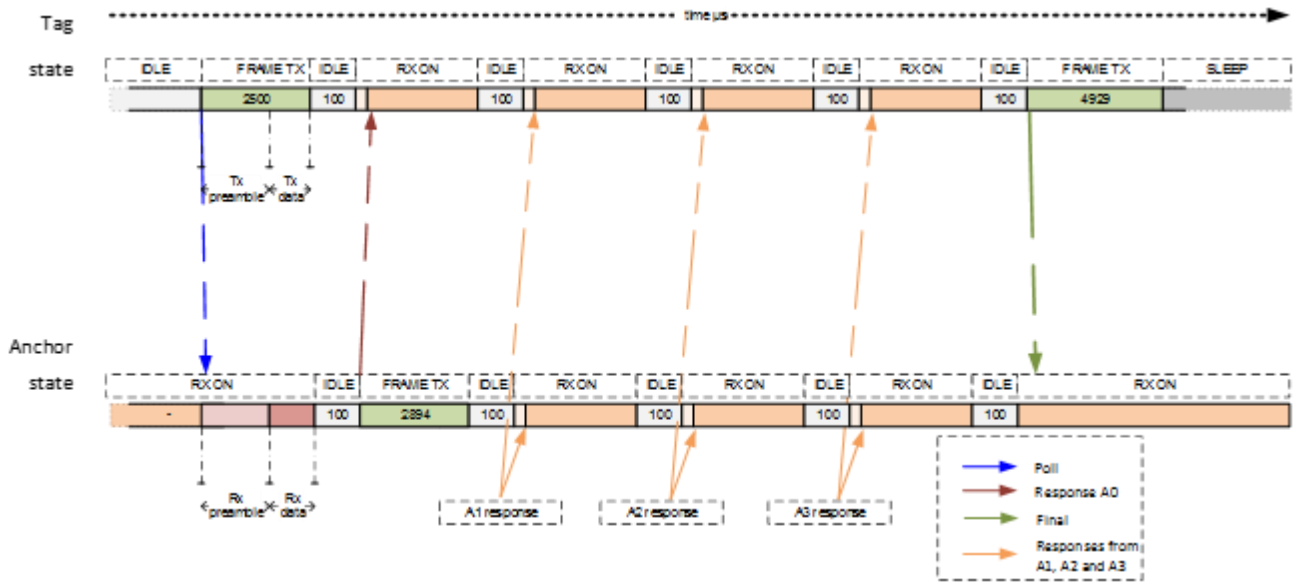

Figure 10: TWR timing profile

In this TREK demo delayed response time are used, as shown in Figure 10. The response times vary depending on the anchor ID. They are kept to a minimum, i.e. the anchors will try and reply after Poll reception as soon as possible. The turnaround time is limited by the SPI frequency and microprocessor event processing time. This means that the total time of ranging exchange takes about 26 ms in 110 kbps mode and 2.25 ms in 6.81 Mbps mode.

Table 7: Frame timings

Message	Bytes	Data Rate	Preamble (sym)	SFD (sym)	Frame duration (µs)
Poll	13	110 kbps	1024	64	2500
Response	16	110 kbps	1024	64	2894
Final	44	110 kbps	1024	64	4929
Poll	13	6.81 Mbps	128	8	176
Response	16	6.81 Mbps	128	8	182
Final	44	6.81 Mbps	128	8	214

4.6 Location rates

The TREK1000 supports 2 location rates as shown in Table 8. As auto-positioning is used the last two slots are taken by anchor-to-anchor two-way-ranging, so a maximum possible number of tags supported is 8. The slot times are derived from the times as shown Table 8 with some guard times added.

Table 8: Location rates

Mode	Data Rate	N Slots	T_s (ms)	T_{sf} (ms)	Hz
2, 4	6.81 Mbps	10	10	100	10
1, 3	110 kbps	10	28	280	3.5

5 BUILDING AND RUNNING THE CODE

5.1 External Libraries

The DecaRanging ARM application consists of STM Libraries and Decawave application and driver sources. All of these are provided in the zip of the source code. There are two zips of the code provided, one is for building the code with CooCox IDE and the other for using the ST System Workbench IDE. The user has a choice of which one they would like to use. In either case they just need to unzip the source and open the relevant project file *DecaRanging.coproj* (if CooCox IDE has already been installed; see paragraph below if not) or import "existing projects into workspace" if using ST System Workbench (AC6 – elipse) IDE. For installation of ST System Workbench – please read ST Installation Guide [5]

5.2 Building the code with Coocox IDE

As an example development environment, this code can be built using CooCox IDE. This code building guide assumes that the reader has ARM Toolchains installed and is familiar with building code using the CooCox IDE. In the DecaRangeRTLS ARM software project we use the GNU Tools ARM for Embedded toolchain. GNU Tools ARM for Embedded can be found at: https://launchpad.net/gcc-arm-embedded

CooCox IDE can be downloaded from: http://www.coocox.org/software.html. Please follow the "Read More" link and download version 1.7.8. The released code was built using version 1.7.8.

5.3 Building configuration options

The example application has a couple of different build configuration options which can be exercised (see instance.h and port.h for more details):

```
#define DEEP_SLEEP     (0) //To disable deep-sleep in the tag set this to 0

#define CORRECT_RANGE_BIAS  (1) // To remove compensation for small bias due to uneven
accumulator growth at close up high power set this to 0

#define ANCTOANCTWR (1) // To disable anchor to anchor ranging this should be set to 0

#define USB_SUPPORT // To disable USB Virtual COM port feature this line should be removed

#define LCD_UPDATE_ON (1) // To stop updating LCD during ranging this should be set to 0
```

6 OPERATIONAL FLOW OF EXECUTION

6.1 Introduction

This section is intended to be a guide to the flow of execution of the software as it runs, reading this and following it at the same time by looking at the code should give the reader a good understanding of the basic way the software operates as control flows through the layers to achieve transmission and reception. This understanding should be an aid to integrating/porting the ranging function to other platforms.

To use this effectively, the reader is encouraged to browse the source code (e.g. in the ST Systemworkbench or CooCox IDEs) at the same time as reading this description, and find each referred item in the source code and follow the flow as described here.

6.2 The main application entry

The application is initialised and run from the *src/application/dw_main()*. Firstly we initialise the HW and various ARM microcontroller peripherals, *peripherals_init()* and *spi_peripheral_init()* functions are used for this also LCD is initialised (*initLCD()*). In the Cube Mx project this is done in *main.c*, and these are auto generated by the Cube MX application. The instance roles (Tag or Anchor) and channel configurations (channel, PRF, data rate etc.) are set up by a call to *inittestapplication()* function. Finally the *tag_run()* or *anch_run()* is called periodically from *while(1)* loop which runs the instance state machines described below. In parallel the DW1000 interrupt line is enabled so any events (e.g. transmitted frames or received frames) are processed in the *dwt_isr()* call (in *src/platform/port.c*).

If there is a new range calculated or ranging report received (*instance_newrange()*), the application will prepare output buffer to be sent over the Virtual COM port, and update the LCD display.

The *usb_run()* is also called from the *while(1)* loop, it processes any data sent to the application over USB/Virtual COM port and outputs any data present in the tx_buff[] (*send_usbmessage()*).

6.3 Instance state machine

The instance state machine delivers the primary DecaRangeRTLS function of range measurement. The instance state machine together with the RX and TX callback functions performs the interleaved two-way ranging by forming the messages for transmission (TX), commanding their transmission, by commanding the receive (RX) activities, by recording the TX and RX timestamps, by extracting the remote end's TX and RX timestamps from the received *Final* messages, and, by performing the time-of-flight calculation.

The instance code is invoked using the function *tag_run()* for a tag and *anch_run()* for an anchor, the paragraphs below trace the flow of execution of this instance state machine from initialisation through the TX and RX operations of a ranging exchange. This is done primarily by looking at the operation of the Tag end. It starts by sending a *Poll* message, awaits a *Responses* (up to 4 *Responses* can be received) and then sends the *Final* message to complete the ranging exchange. On the reception of the Final each anchor can calculate the ToF which will be sent in the next *Response* message.

The anchor transitions are not discussed in detailed here, but after reading the description of tag execution flow below the reader should be well equipped to similarly follow the anchor flow of execution.

The *tag_run()* function is the main function for the tag instance; it should be run periodically. It checks if there are any outstanding events. Once the interrupt happens relevant callback function is called *rx_ok_cb_tag(), rx_to_cb_tag(), rx_err_cb_tag()* or *tx_conf_cb_tag()* and acted upon. Some events will schedule a response or re-enable the receiver, but also read the relevant TX/RX data from the DW1000 and queued it up in the event queue so that the application can process it in the background. The application internal timers are also checked in the *tag_run()* function (e.g. Sleep timer). Below paragraphs describe the *tag_app_run ()* sate machine in detail:

6.3.1 Initial state: TA_INIT

Function *tag_app_run ()* contains the state machine that implements the tag side of the two-way ranging function, the part of the code executed depends on the state and is selected by the "`switch (inst->testAppState)`" statement at the start of the function. The initial state "`case TA_INIT`"[1] performs initialisation and determines the next state to run. In the case of a tag we want to go to Sleep state after initialisation, and then wake up and start the ranging exchange, thus the state "`inst->nextState`" is changed to "`TA_TXPOLL_WAIT_SEND`" and "`inst->instToSleep`" is set to TRUE.

6.3.2 State: TA_SLEEP_DONE

In this state the microprocessor will wake up the DW1000 from DEEP SLEEP once the sleep timeout expires. After waking up any of the DW1000 registers that are not preserved will be re-programmed and the state will change to `inst->testAppState = inst->nextState;` which will be "`TA_TXPOLL_WAIT_SEND`".

Note: In order to minimise power, the microprocessor uses DW1000 RSTn pin to notify when the DW1000 enters the INIT mode after wake up and is ready for operation. This minimises the time microprocessor would otherwise wait before polling to check that DW1000 has entered INIT state. Before reading or writing over SPI the micro needs to make sure the DW1000 is in IDLE, the time to IDLE will take 35 µs. The wake up function is *port_wakeup_dw1000_fast()*.

6.3.3 State: TA_TXPOLL_WAIT_SEND

In the state "`case TA_TXPOLL_WAIT_SEND`", we want to send the *Poll* message, so firstly we set up the destination address and all the other parameters/bytes of the *Poll* message. The *Poll* message is a broadcast message as the destination address is set to 0xffff.

The *tag_app_run ()* state machine state is set to "`TA_TX_WAIT_CONF`", and as that state has more than one use, "`inst->previousState = TA_TXPOLL_WAIT_SEND`" is set to as a control variable.

Note: In the case if a tag sending the *Poll* message, this message is sent immediately (by a call to *dwt_starttx)* and tag's *Final* message is sent with a delayed send command (state "`case TA_TXFINAL_WAIT_SEND`" as described in section 6.3.8 below), it is required to send the message at an exact

[1] The "TA_" prefix is because these are states in the "Test Application".

and specific time with respect to the arrival of the message soliciting the response. To do this we use delayed send. This is selected by the "`delayedTx`" second parameter to function *instance_send_delayed_frame ()*.

We also configure and enable the RX frame wait timeout, so that if the response is not coming, the tag times-out and restarts the ranging. Also, a receiver is turned on automatically (DWT_RESPONSE_EXPECTED) with a delay, this is because the *Response* is expected after a certain time after the *Poll* transmission is complete so turning on the receiver too early would only waste power.

6.3.4 State: TA_TXE_WAIT

This is the state for the tag which is called before the next ranging exchange starts (i.e. before the sending of next *Poll* message). Here we check if tag needs to enter DEEP_SLEEP mode before the next *Poll* is sent, and call *dwt_entersleep()* if sleep is required.

Note: To save power the tag in TWR RTLS system, a tag will poll and range with a number of anchors and then enter a sleep mode before starting the process again.

6.3.5 State: TA_TX_WAIT_CONF

In the state "`case TA_TX_WAIT_CONF`", we await the confirmation that the message transmission has completed. When the IC completes the transmission a "TX done" status bit is picked up by the device driver interrupt routine which generates an event which is then processed by the TX callback function (*tx_conf_cb_tag()*). The instance, after a confirmation of a successful transmission, will read and save the TX time and calculate "`delayedReplyTime`" which is when we should send the *Final* message to complete the ranging exchange. and then proceed to the next state (`TA_RX_WAIT_DATA`). The next state is thus set "`inst->testAppState = TA_RX_WAIT_DATA`". See 6.3.7 for details of what this does.

6.3.6 State: TA_RXE_WAIT

This is the pre-receiver enable state. Here the receiver is enabled and the instance will then proceed to the `TA_RX_WAIT_DATA` where it will wait to process any received messages or will timeout. Sometimes this state is skipped if the receiver is turned on automatically (as we had `DWT_RESPONSE_EXPECTED` set as part of TX command). We use automatic delayed turning on of the receiver as we know the exact times the responses are sent, as they are using delayed transmissions. This it is possible (and desirable for power efficiency) to delay turning on the receiver until just before the response is expected. (Delayed RX is not part of the IEEE standard primitive but is an extension to support this DW1000 feature). The next state is: "`inst->testAppState = TA_RXE_WAIT_DATA`".

Note: If a delayed transmission fails the transceiver will be disabled and the receiver will then be enabled normally in this state.

6.3.7 State: TA_RX_WAIT_DATA

The state "`case TA_RX_WAIT_DATA`" (in *instance_tag.c*) handles all the RX messages expected for the tag application, and the corresponding states for an anchor are found in *instance_anch.c*. We "`switch (message)`", and handle message arrival as signalled by a received event. If a good frame has been received

(SIG_RX_OKAY), we firstly check which addressing mode is used (short or long or both) and then we look at the first byte of MAC payload data (beyond the IEEE MAC frame header bytes) and "switch(rxmsg->messageData[FCODE])". FCODE is a Decawave defined identifier for the different DecaRanging messages; see Figure 9, for details. For the point of view of the discussions here the tag is awaiting the anchor's response message so we would expect the FCODE to match "RTLS_DEMO_MSG_ANCH_RESP". We note the RX timestamp of the message "anchorRespRxTime". If it is time to send the Final (e.g. if this was the 4th response message) then our next states is set to: `inst->testAppState = TA_TXFINAL_WAIT_SEND ; // then send the final response`

The state "case TA_RX_WAIT_DATA" also includes code to handles the "SIG_RX_TIMEOUT" message, for the case where the expected message does not arrive and the DW1000 triggers a frame wait timeout event. The DW1000 has an RX timeout function to allow the host wait for IC to signal either data message interrupt or no-data timeout interrupt[2]. When the timeout happens, the tag will go back to restart the ranging exchange or send a Final if there were any Responses received.

6.3.8 State: TA_TXFINAL_WAIT_SEND

In the state "case TA_TXFINAL_WAIT_SEND", we want to send the *Final* message.

The *Final* message includes embedded the TX time-stamp of the tag's poll message "inst->tagPollTxTime" along with the four RX time-stamps of the anchors response messages, and embedded predicted (calculated) TX time-stamp for the final message which includes adding the antenna delay "inst->txantennaDelay". A mask which shows which response times are valid is also inserted into the *Final* message.

The final message is sent at a specific time with respect to the transmission of the Poll, this is done using delayed send, selected by the "delayedTx" second parameter to function *instance_send_delayed_frame ().*

We finish the processing by setting control variable "inst->previousState = TA_TXFINAL_WAIT_SEND" to indicate where we are coming from and we set the "inst->testAppState = TA_TX_WAIT_CONF" selecting this as the new state for the next call of the *tag_app_run ()* state machine.

6.3.9 State: TA_TX_WAIT_CONF (for *Final* message TX)

In the state "case TA_TX_WAIT_CONF", (as detailed in section 6.3.5) we await the confirmation that the message transmission has completed. Here we will set "inst->testAppState = TA_TXE_WAIT", and then go to Sleep.

6.3.10 CONCLUSION

The above should be enough of a walkthrough of the state machine that the reader should be able to decipher the anchor activity (and any remaining activity of tag).

In summary the anchor waits indefinitely in the state "case TA_RX_WAIT_DATA" until it receives a *Poll* message. Once it receives the poll it starts the ranging exchange and finishes with a calculation of TOF

[2] This idea here (although no code is yet written for this) is to facilitate the host processor entering a low power state until awakened by either the RX data arriving or the no data timeout.

(range) report, which it reports to the LCD/USB and also, sends back to the tag in the next *Response* message.

The Anchor ID 0, (section 2.4.2 Operating mode – Anchor) will also calculate the correct sleep delay correction to send back to the tag so that the next ranging exchange starts in the correct superframe slot.

7 REFERENCES

Ref	Author	Title
[1]	Decawave	DW1000 Data Sheet
[2]	Decawave	DW1000 User Manual
[3]	Decawave	TREK1000 User Manual
[4]	IEEE	IEEE 802.15.4-2011 or "IEEE Std 802.15.4™-2011" (Revision of IEEE Std 802.15.4-2006). IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Computer Society Sponsored by the LAN/MAN Standards Committee. Available from http://standards.ieee.org/
[5]	Decawave/ST	01_Installation of the tools and drivers.pdf

8 DOCUMENT HISTORY

Table 9: Document History

Revision	Date	Description
1.1	31st March 2015	Initial release for production device.
2.0	30th September, 2015	Scheduled update
2.1	30th October, 2015	Update to include changes to slot / superframe timings
2.2	15th November, 2017	Updated to reflect the changes in the source code due to refactoring and any other updates as a result of porting to Cube Mx/ST System workbench project and IDE.

9 MAJOR CHANGES

Release 2.0

Page	Change Description
All	Update of version number to 2.0
All	Various typographical changes
Sections 2, 3, 4 and 6	Updated the doc to the new TWR scheme (asymmetric, interleaved)

Release 2.1

Page	Change Description
All	Update of version number to 2.1
All	Various typographical changes

Sections 2.4.1	Updated the slot/superframe timings for 6.81 Mbps rate (10 slots of 10 ms are now used), the location rate of 10 Hz remains the same.

Release 2.2

Page	Change Description
All	Update of version number to 2.2
All	Various typographical changes
Sections: 2, 5 and 6	Updated to reflect the changes in the source code due to refactoring and any other updates as a result of porting to Cube Mx/ST System workbench project and IDE.

Release 2.3

Page	Change Description
All	Update with new Logo

10 FURTHER INFORMATION

Decawave develops semiconductors solutions, software, modules, reference designs - that enable real-time, ultra-accurate, ultra-reliable local area micro-location services. Decawave's technology enables an entirely new class of easy to implement, highly secure, intelligent location functionality and services for IoT and smart consumer products and applications.

For further information on this or any other Decawave product, please refer to our website www.decawave.com.