SOURCE CODE GUIDE

DECARANGERTLS ARM SOURCE CODE

Understanding and using the DecaRangeRTLS ARM source code

Version 1.00

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,
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.

Appendix A – Operational flow of execution 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 1.05"` application version and `"DW1000 Device Driver Version 02.15.01"` driver version. The device driver version information may be found in source code file "deca_version.h", and the application version is specified in "main.c".

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

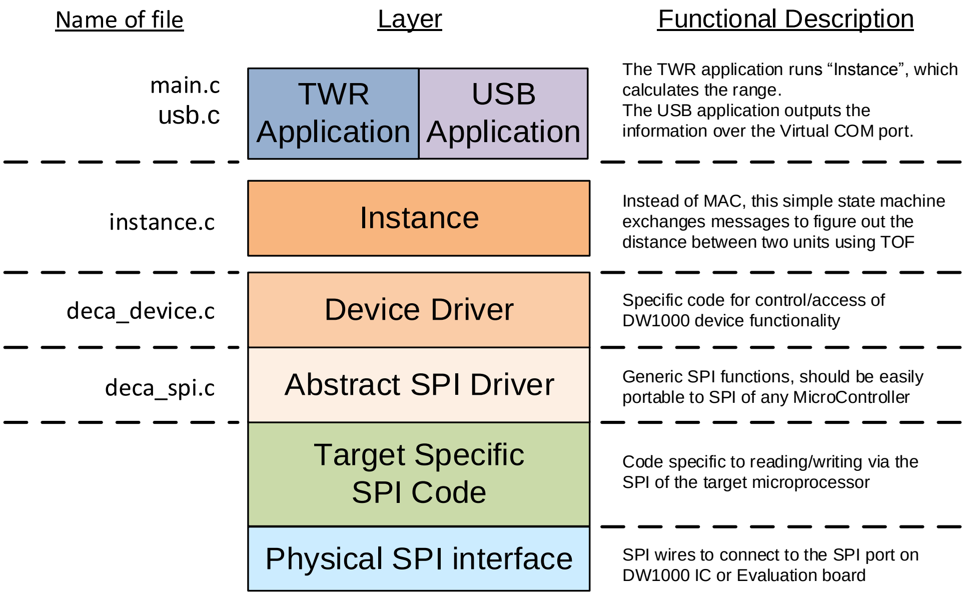

Name of file
Layer
Functional Description
main.c
usb.c
TWR Application
USB Application
The TWR application runs "Instance", which calculates the range.
The USB application outputs the information over the Virtual COM port.
instance.c
Instance
Instead of MAC, this simple state machine exchanges messages to figure out the distance between two units using TOF
deca_device.c
Device Driver
Specific code for control/access of DW1000 device functionality
deca_spi.c
Abstract SPI Driver
Generic SPI functions, should be easily portable to SPI of any MicroController
Target Specific SPI Code
Code specific to reading/writing via the SPI of the target microprocessor
Physical SPI interface
SPI wires to connect to the SPI port on DW1000 IC or Evaluation board

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*

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 *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 instance.c) provides a simple ranging demonstration application. This instance code sits where the MAC would normally reside. For expediency in developing the ranging demonstration to showcase ranging and performance of the DW1000, the ranging demo application was implemented directly on top of the DW1000 driver API.

The two-way ranging RTLS demo application is implemented by the state machine in function *testapprun()*, called from function *instance_run()*, which is the main entry point for running the instance code. 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. A separate DecaRangeRTLS PC application can then use the ranges from multiple tag-anchor pairs and output Tag's location.

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 it will wake up and range to 4 anchors again. The tag ranges to Anchor 0 (gateway anchor) first, then it ranges to anchors 1, 2 and 3 in turn. All of the range results are reported via the USB port. The scheduling period is 1024 ms. This is the period from the time that the Tag sends its first poll to anchor #0, through polling and ranging with all the (up to 4) anchors present, through its sleeping period and back to the same instant of sending the first poll to the anchor #0 again.

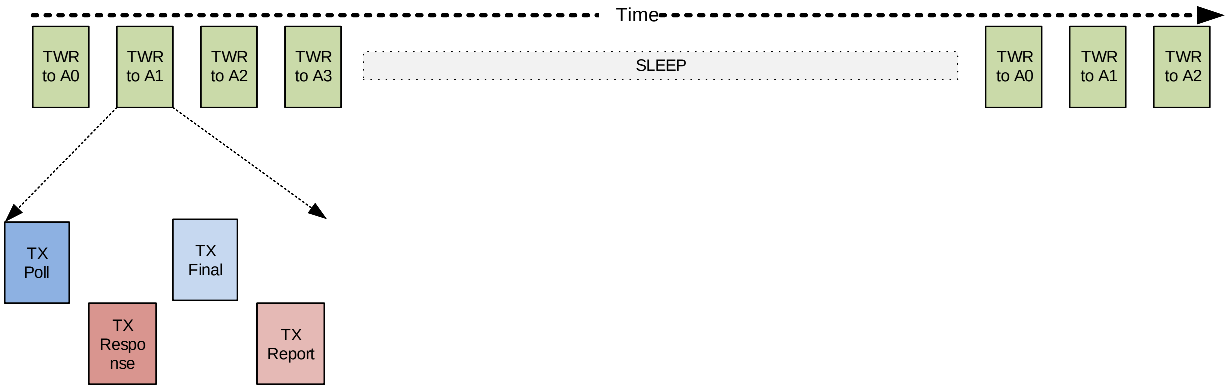


Figure 2: Tag TWR RTLS time profile

If the ranging to A0 is unsuccessful, i.e. if there is no response to the Poll message, the Tag will send a second Poll, if there is still no response then the Tag will go to Sleep. It will not attempt to range to other 3 anchors. The same is true if the ranging to A1 is unsuccessful, the tag will not attempt to range to A2 or A3.

To support multiple Tags (without interfering), we are using a TDMA approach. The time period (1024ms) which we call a *superframe* is divided into slots. There are 8 slots in the superframe, so each slot is 128ms and, we assign one slot to each tag, so it can support a maximum of 8 tags.

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

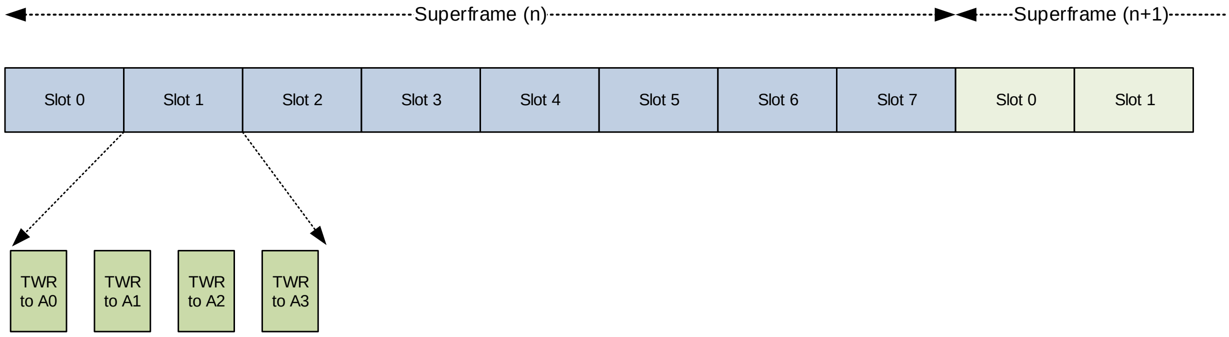


Figure 3: Superframe structure

2.4.2 Gateway Anchor

Every time the gateway anchor receives a Poll from a tag, it checks whether if that poll is being transmitted in the correct slot position. The gateway anchor sends a correction factor to the tag (in its Response message) that is used to correct the next time of transmission for the tag. This sets and maintains the tag transmissions into the correct slot position. In TREK there are 8 tags, define by the 8 tag address selection switches, and the slot assignments are directly corresponded with the tag address, (tag 0 in slot 0, tag 1 in slot 1, etc.).

2.4.3 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 Ranging Report message.

The gateway anchor will report ranges it calculates for the tags that range with it, but also any range reports it receives from other anchors. It does not use frame filtering so it can receive any messages on the air.

The other anchors (A1, A2 and A3) will each only report the range results they calculate for the tags that range with them alone. They have frame filtering enabled so they do not receive any messages other than those directly addressed to them.

When in Listener mode the unit will report any ranges it receives in ranging reports. It will not be involved in any ranging exchange but, as it has no frame filtering enabled, will hear reports that are on the air.

2.4.4 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 4 shows the arrangement and general operation of the two-way ranging as implemented by the DecaRangeRTLS ARM application.

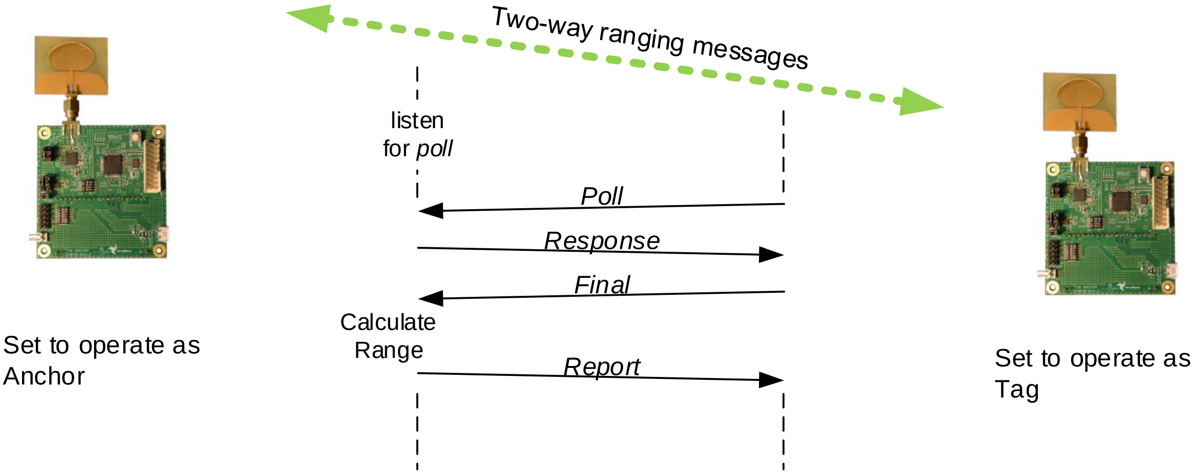


Figure 4: Two way ranging in DecaRangeRTLS ARM

Above this instance level is the application that provides the user interface described in section 2.5 below. In addition to the *instance_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_run(), instance_close(), instancedevicesoftreset(), instancegetrole(), instancesetrole(), instancesetantennadelays(), instancereaddeviceid(), instance_readaccumulatordata(), instancesetreporting(), instancegetcurrentrangeinginfo(), instancesetaddresses(), instancesettagsleepdelay(), instancesetreplydelay(), instancesetspibitrate(), instance_rxcallback(), instance_txcallback(), inst_processrxtimeout().

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

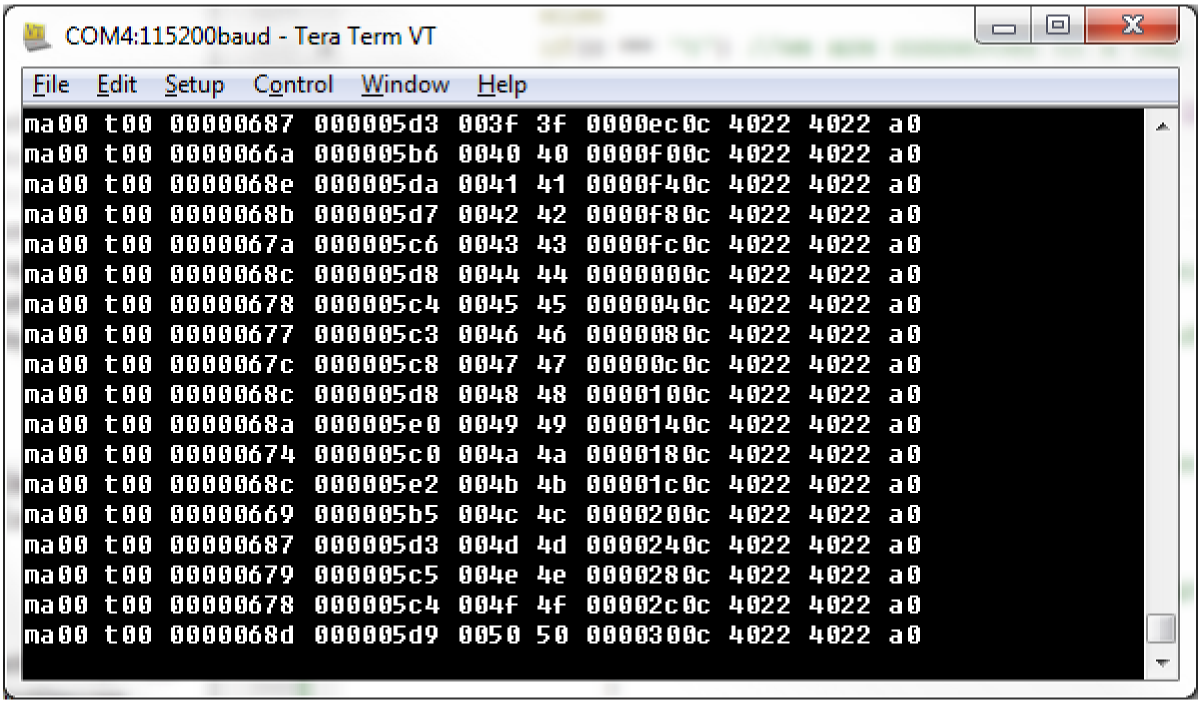
The ability of the application layer to display results depends on the capability of the hardware platform. On the EVB1000 evaluation board the LCD is used to display the resultant range from a range measurement.

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 5** shows example output from anchor 0 as viewed on Teraterm terminal emulator (communication program).


COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
ma00 t00 00000687 000005d3 003f 3f 0000ec0c 4022 4022 a0
ma00 t00 0000066a 000005b6 0040 40 0000f00c 4022 4022 a0
ma00 t00 0000068e 000005da 0041 41 0000f40c 4022 4022 a0
ma00 t00 0000068b 000005d7 0042 42 0000f80c 4022 4022 a0
ma00 t00 0000067a 000005c6 0043 43 0000fc0c 4022 4022 a0
ma00 t00 0000068c 000005d8 0044 44 0000000c 4022 4022 a0
ma00 t00 00000678 000005c4 0045 45 0000040c 4022 4022 a0
ma00 t00 00000677 000005c3 0046 46 0000080c 4022 4022 a0
ma00 t00 0000067c 000005c8 0047 47 00000c0c 4022 4022 a0
ma00 t00 0000068c 000005d8 0048 48 0000100c 4022 4022 a0
ma00 t00 0000068a 000005e0 0049 49 0000140c 4022 4022 a0
ma00 t00 00000674 000005c0 004a 4a 0000180c 4022 4022 a0
ma00 t00 0000068c 000005e2 004b 4b 00001c0c 4022 4022 a0
ma00 t00 00000669 000005b5 004c 4c 0000200c 4022 4022 a0
ma00 t00 00000687 000005d3 004d 4d 0000240c 4022 4022 a0
ma00 t00 00000679 000005c5 004e 4e 0000280c 4022 4022 a0
ma00 t00 00000678 000005c4 004f 4f 00002c0c 4022 4022 a0
ma00 t00 0000068d 000005d9 0050 50 0000300c 4022 4022 a0
```

Figure 5: Example Teraterm window showing the debug info sent via COM port

The format of ranging report message as sent over the USB port is:
"ma00 t00 range rawrange rangenum rangeseq rangetime txad rxad zZ"

Each ranging report message starts with "m".

a00 this is the Anchor ID, 0x00 is used for Gateway, 0x01 is A1, 0x02 is A2 and 0x03 is A3.

t00 this is the Tag ID (range is 0x00 to 0x00)

range this is range in mm, it is bias corrected 32-bit hex number of the calculated range between A and T as reported in the 1st two fields.

rawrange this is a raw range in mm, 32-bit hex number of the calculated range between A and T as reported in the 1st two fields.

rangenum this is a 16-bit hex number of the number of ranges since the unit has been powered on

rangeseq this is the range sequence number (modulo 256)

rangetime this is the time of receiving a range report at Tag or calculation the ToF at Anchor, units are ms (local Anchor/Tag MCU counter), (32 bit hex number)

txad the TX antenna delay

rxad the RX antenna delay

zZ this is the ID of the Tag or Anchor or Listener the PC terminal is connected to
(z = "t" or "a or "l")
(Z = 0, 1, 2, 3)

2.7 *Complete list of source code files*

Table 2 gives a list of the 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
port.c	ARM peripheral and GPIO configuration
port.h	ARM peripheral and GPIO configuration definitions
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.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.
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 4 anchors and then go into DEEP SLEEP mode. (It will be woken up after 1024 ms 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.

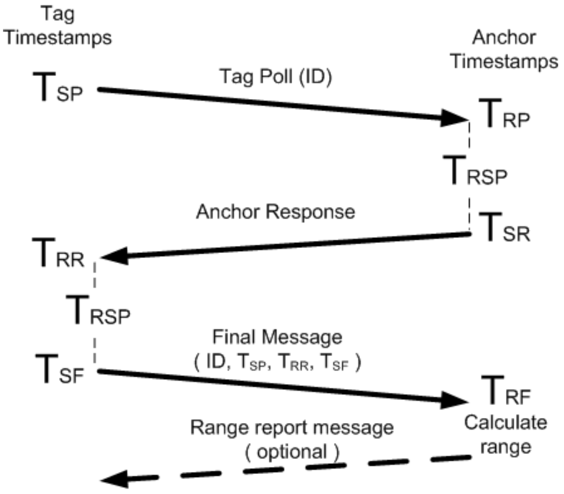
- The tag sends a *Poll* message addressed to the target anchor and notes the send time, T_{SP}. The tag listens for the *Response* message. If no response arrives after some period the tag will resend the *Poll* message and if still no *Response* it will time out and go to Sleep.

- The anchor listens for a *Poll* message addressed to it. When the anchor receives a poll it notes the receive time T_{RP}, and sends a *Response* message back to the tag, noting its send time T_{SR}.

- When the tag receives the *Response* message it notes the receive time T_{RR} and sets the future send time of the *Final* response message T_{SF}, (a feature of DW1000 IC), it embeds this time in the message before initiating the delayed sending of the *Final* message to the anchor.

- The anchor receiving this *Final* response message now has enough information to work out the range.

- 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 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 6 shows this exchange and gives the formula used in the calculation of the range.



The tag sees Round Trip Delay time T_{TRT} of (T_{RR} - T_{SP})
The anchor sees Round Trip Delay time T_{ART} of (T_{RF} - T_{SR})

After receiving the Final message, the anchor knows all the timestamps, so it can:
(a) remove its response time: (T_{SR} - T_{RP}) from the Tag's T_{TRT}, and
(b) remove tags response time: (T_{SF} - T_{RR}) from Anchor's T_{ART},
to give antenna to antenna round trip times

The anchor then averages these two resultant round trip times to remove the effects of each end's clock frequency differences, (a technique which works best when the RX-to-TX response T_{RSP} is the same at both ends), and divides by 2 to get a one-way trip time.

Putting all this into one equation gives a Time of Flight:
$((T_{RR} - T_{SP}) - (T_{SR} - T_{RP}) + (T_{RF} - T_{SR}) - (T_{SF} - T_{RR})) \ / \ 4$

or $TOF = (2T_{RR} - T_{SP} - 2T_{SR} + T_{RP} + T_{RF} - T_{SF}) / 4.$

Multiplying the TOF by c, the speed of light (and radio waves), gives the distance (or range) between the two devices.

Figure 6: Range calculation in DecaRanging

- After this the anchor turns on its receiver again to await the next poll message, while the tag will attempt to range to other 3 anchors, and once finished (or if any range exchange fails) it will go into DEEP SLEEP mode.

3.2 *Practical considerations for ranging in a real product*

The application note “*Moving from TREK to a (commercial TWR RTLS) product” to a 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.

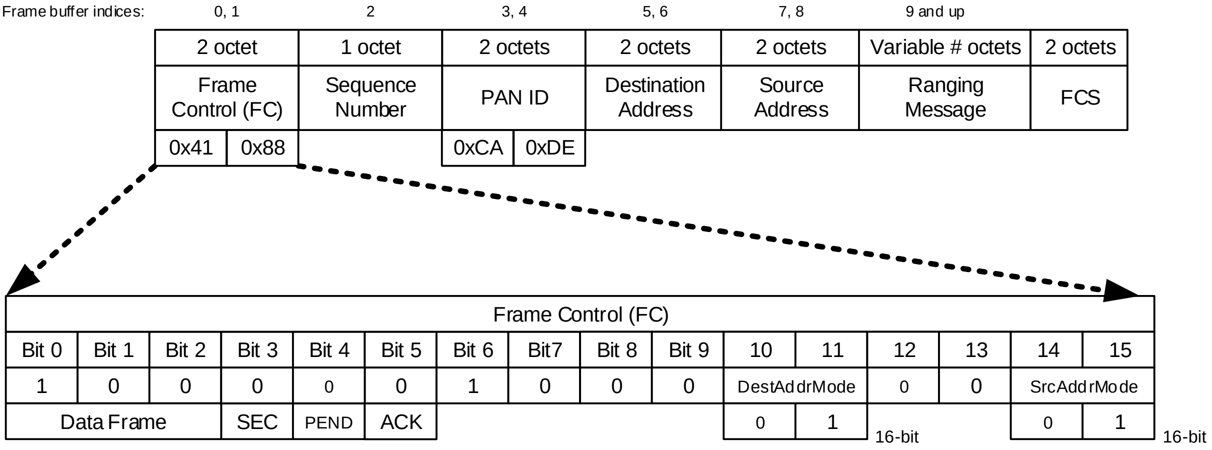
The EVB1000 units as part of the TREK1000 kit have the antenna delays calibrated, and programmed into the DW1000 OTP (one-time-programmable) memory. However if DecaRangeRTLS SW is downloaded on an EVB1000 which has not been calibrated for TREK, the application will use the default antenna delay value as set in the instance_calib.c file, there are two values, one for each channel option (2/5) **514.83** ns and **514.65** ns respectively. The value specified is divided equally between TX and RX antenna delays. The default value has been experimentally set by adjusting it until the reported distance averaged to be the measured distance. The need to re-tune the Antenna Delay is discussed in section 5.1 below.

4 MESSAGES USED IN TREK TWR

As shown in **Error! Reference source not found.**, four messages are employed in the two-way ranging: the poll essage, the response message, the final message, and the range report 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 7 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.



Frame buffer indices:	0, 1	2	3, 4	5, 6	7, 8	9 and up	
	2 octet	1 octet	2 octets	2 octets	2 octets	Variable # octets	2 octets
	Frame Control (FC)	Sequence Number	PAN ID	Destination Address	Source Address	Ranging Message	FCS
	0x41 0x88		0xCA 0xDE				

Frame Control (FC)																
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit7	Bit 8	Bit 9	10	11	12	13	14	15	
1	0	0	0	0	0	1	0	0	0	DestAddrMode		0	0	SrcAddrMode		
Data Frame			SEC	PEND	ACK					0	1	16-bit		0	1	16-bit

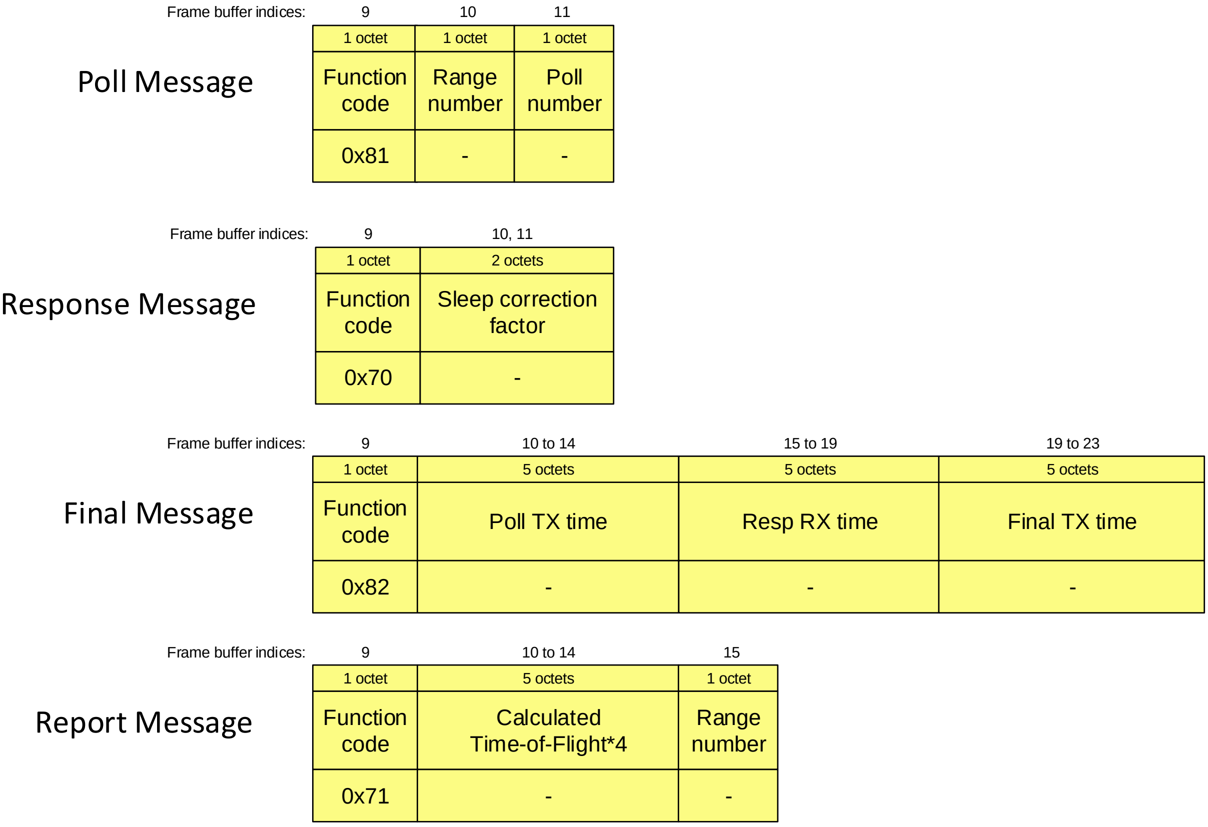
Figure 7: General ranging frame format

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 8 and described in sections 4.2 to 4.5 below. 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 7 to form the complete ranging message in each case.



Poll Message

Frame buffer indices:	9	10	11
	1 octet	1 octet	1 octet
	Function code	Range number	Poll number
	0x81	-	-

Response Message

Frame buffer indices:	9	10, 11
	1 octet	2 octets
	Function code	Sleep correction factor
	0x70	-

Final Message

Frame buffer indices:	9	10 to 14	15 to 19	19 to 23
	1 octet	5 octets	5 octets	5 octets
	Function code	Poll TX time	Resp RX time	Final TX time
	0x82	-	-	-

Report Message

Frame buffer indices:	9	10 to 14	15
	1 octet	5 octets	1 octet
	Function code	Calculated Time-of-Flight*4	Range number
	0x71	-	-

Figure 8: 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.
3	-	Poll Number: After sending a 1st poll, if there is no response the Tag will send a 2nd poll. This field is set to 0 or 1, to indicate whether the poll is the 1st or 2nd ranging attempt.

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.

4.4 *Final message*

The final message is sent by the tag after receiving the anchor's response message. The final message is 16 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 to 6	-	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.
7 to 11	-	Resp RX time: This 5 octet field is the RX timestamp for the response, i.e. the time the tag received the response frame from the anchor.
12 to 16	-	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).

4.5 *Range report message*

Upon receiving the *Final message* the anchor can then calculate the time of flight distance to the tag. The range report message sends the result over-the-air back to the tag unit. Table 6 lists and describes the individual fields within the range report message.

Table 6 – Fields within the range report message

Octet #'s	Value	Description
1	0x71	Function code: This octet identifies the message as a range report
2 to 6	-	Time of flight: This five octet field is four times the time-of-flight between the tag and anchor. This is divided by 4 and multiplied by the speed of radio waves to convert it to a distance, before reporting it through the USB.
7	-	Range number: This is the parameter value that was in the poll message, see Table 3.

As well as sending the ranging report to the tag, each anchor also reports the ranging result via its USB port. The range report message serves to give the ranging result back to the tag, which also reports it out its USB port. Gateway anchor #0, (alone of all the anchors), also listens for ranging reports sent by other anchors and reports these via its USB port.

4.6 *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 = 14, Response = 15, Final = 27, Report = 18. If we take the longest message, the total frame duration is 3.423 ms for the 110 kbps mode and 0.194 ms for the 6.81 Mbps mode.

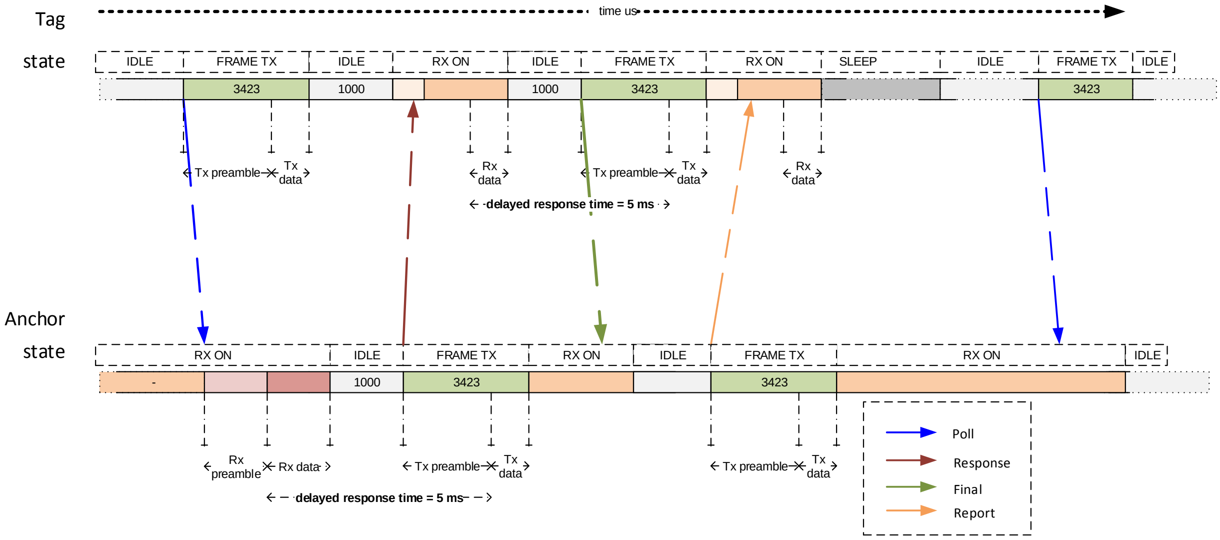


Figure 9: TWR timing profile

In this TREK demo delayed response time of 5 ms is used, as shown in Figure 9, this means that the total time of ranging exchange is 16 ms (where no retry of the Poll is necessary), and the time to range to all four anchors is thus 64 ms. The worst case here is about 80 ms, which only happen if the 1st poll to each anchor fails and is retried, since each roll retry adds about 4 ms.

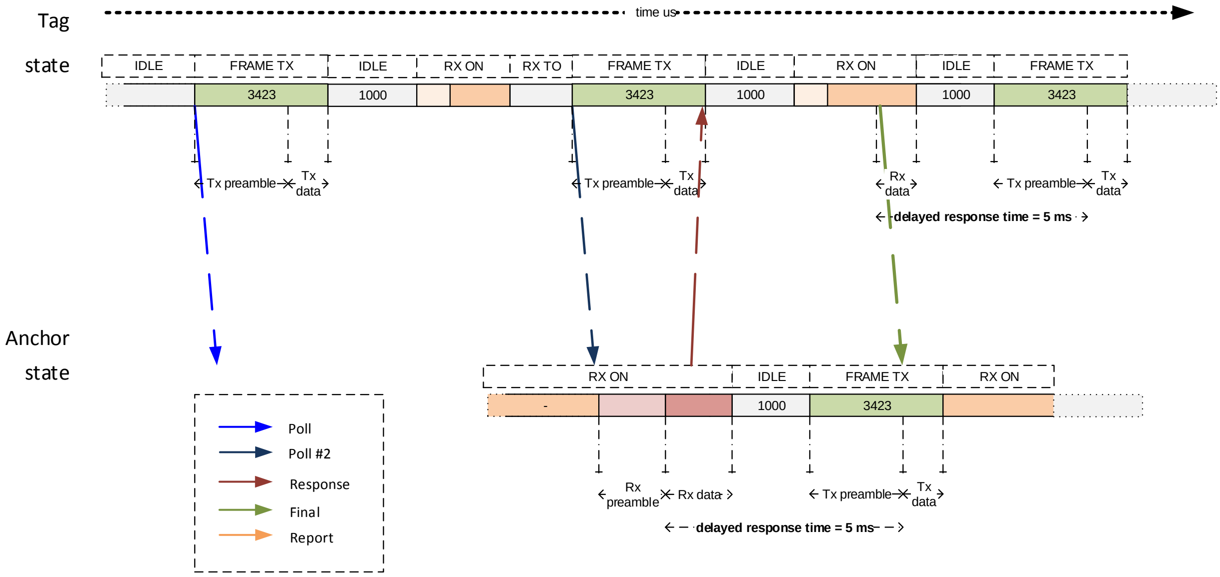


Figure 10: TWR when 1st Poll does not get Response

Figure 10 shows a timing diagram when there is no Response to the 1st Poll message, the Tag will send a 2nd Poll to the same Anchor. If this fails the Tag will stop with ranging exchange and will go to Sleep.

In the case of the 6.81 Mbps mode the frame durations are shorter but the turn-around times are not optimised, (i.e. have been left at 5ms) so a single exchange is completed in 10 ms.

Customers working with the source code and using the 6.81 Mbps mode might consider shortening the response times to handle a higher tag density and/or save some power. DecaWave's TWR application note may be consulted for additional details on this.

5 BUILDING AND RUNNING THE CODE

5.1 *External Libraries*

The DecaRangeRTLS ARM application consists of STM Libraries and DecaWave application and driver sources. All of these are provided in the zip of the source code. The user just needs to unzip the source and open the project file *DecaRangeRTLS_ARM.coproj* (if CooCox IDE has already been installed, paragraph below explains how to install CooCox IDE).

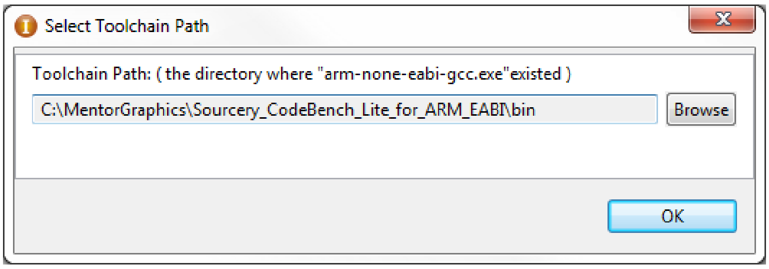
5.2 *Building the code*

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 *Mentor Graphic*s toolchain (*Sourcery CodeBench Lite Edition*).

CooCox IDE can be downloaded from: http://www.coocox.org/software.html.

Mentor Graphich toolchain can be downloaded from: http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/overview.

A user can also select a different toolchain:



Toolchain Path: (the directory where "arm-none-eabi-gcc.exe"existed)

C:\MentorGraphics\Sourcery_CodeBench_Lite_for_ARM_EABI\bin

Figure 11: Select toolchain path

5.3 *Building configuration options*

The example application has a couple of different build configuration options (see instance.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
```

6 APPENDIX A – OPERATIONAL FLOW OF EXECUTION

This appendix 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 eclipse IDE) at the same time as reading this description, and find each referred item in the source code and follow the flow as described here.

6.1 *The main application entry*

The application is initialised and run from the *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()*). Then 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 *instance_run()* is called periodically from *while(1)* loop which runs the instance state machine 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.

If there is a new range calculated or ranging report received (*instancenewrange()*), 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[].

6.2 *Instance state machine*

The instance state machine delivers the primary DecaRangeRTLS function of range measurement. The instance state machine does two-way ranging by forming the messages for transmit (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 *testapprun()*, 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 *Blink* message and waiting to receive a *Ranging Initiation* message before starting ranging exchange. Then it will send a *Poll* message, await a *Response* and then send the *Final* message to complete the ranging exchange. If the anchor is sending the ToF report the tag will enable its receiver after the final message to receive the report and display the range.

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 *instance_run()* function is the main function for the instance; it should be run periodically or as a result of a pending interrupt. It checks if there are any outstanding events (once an interrupt happens *instance_txcallback()* or *instance_rxcallback()* is called and the DW1000 TX or RX event is processed and queued up in the event queue) that need to be processed and calls the *testapprun()* function to process them. It also checks if any timers have expired (e.g. Sleep timer). Below paragraphs describe the *testapprun()* sate machine in detail:

6.2.1 Initial state: TA_INIT

Function *testapprun()* contains the state machine that implements 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 depending on whether the "`inst->mode`" is selecting Tag, Anchor or Listener operation. Let's assume it is a tag and follow the execution of the next state. In the case of a tag we want to go to Sleep state after which we'll initiate the ranging exchange, thus the state "`inst->testAppState`" is changed to "`TA_TXPOLL_WAIT_SEND`" and "`inst->instToSleep`" is set to TRUE.

6.2.2 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 us. Here we do 4 dummy reads to make sure DW1000 is in IDLE before writing to it.

6.2.3 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 then we call function *setupmacframedata()*, which sets up the all the other parameters/bytes of the *Poll* message. The tag will range to 4 anchors in turn, and it will also re-try if first *Poll* that is sent does not get a *Response*.

The *testapprun()* 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()*). However in the case of the anchor responses (state "`case TA_TXRESPONSE_WAIT_SEND`" not documented here), and tag's *Final* message (state "`case TA_TXFINAL_WAIT_SEND`" as described in section 6.2.8 below), it is required to send the message at an exact and specific time with respect to the arrival of

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

the message soliciting the response. To do this we use delayed send. This is selected by the "`delayedTx`" second parameter to function *instancesendpacket()*.

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.

6.2.4 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.

6.2.5 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 (*instance_txcallback()*). The instance, after a confirmation of a successful transmission, will read and save the TX time and then proceed to the next state (`TA_RXE_WAIT`). It will only turn on the receiver if `inst->wait4ack` is not set and await a response message. The next state is thus set "`inst->testAppState = TA_RXE_WAIT`". See 6.2.6 below for details of what this does.

6.2.6 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. Since the receiver will be turned on automatically (as we had `DWT_RESPONSE_EXPECTED` set as part of TX command), the state changes to `TA_RX_WAIT_DATA` to wait for the expected response message from the anchor or timeout. 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.

6.2.7 State: TA_RX_WAIT_DATA

The state "`case TA_RX_WAIT_DATA`" is quite long because it handles all the RX messages expected. This is not very robust behaviour. The tag should really only look for the messages expected from the anchor, (and vice versa). 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 8, 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". In this code, we note the RX timestamp of the message "anchorRespRxTime" and calculate "delayedReplyTime" which is when we should send the *Final* message to complete the ranging exchange. In this case our next (and subsequent states) are set to:

```
    inst->testAppState = TA_TXFINAL_WAIT_SEND ; // then send the final response
```

In this application the anchor will send a report message, so that tag should wait for it before sending the next poll.

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.

```
    inst->testAppState = TA_TXE_WAIT ;  // to check if should go to sleep before next ranging
    inst->nextState = TA_TXPOLL_WAIT_SEND ; // then send the poll
```

6.2.8 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 RX time-stamp of the anchors response message "inst->anchorRespRxTime" and the embedded predicted (calculated) TX time-stamp for the final message itself which includes adding the antenna delay "inst->txantennaDelay".

So, now the *Final* message is composed and we call the *setupmacframedata()* function to prepare the rest of the message structure. The final message is sent at a specific time with respect to the arrival of the message soliciting the response, this is done using delayed send, selected by the "delayedTx" second parameter to function *instancesendpacket().*

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 *testapprun()* state machine.

[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.

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

In the state "`case TA_TX_WAIT_CONF`", (as detailed in section 0 above) we await the confirmation that the message transmission has completed. Here we will save the TX time and then proceed to the next state (`TA_RXE_WAIT`). It will only turn on the receiver if `inst->wait4ack` is not set and await a response message. The next state is thus set "`inst->testAppState = TA_RXE_WAIT`".

6.2.10 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 (range) report, which it reports to the LCD/USB and also, sends back to the tag.

The Anchor ID0, which is the gateway anchor (section 2.4.2 Gateway Anchor) will also calculate the correct delay correction to send back to the tag so that the next ranging exchange starts in the correct superframe slot.

7 APPENDIX B – BIBLIOGRAPHY

1	DecaWave DW1000 Datasheet
2	DecaWave DW1000 User Manual
3	EVK1000 User Manual
4	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/

8 ABOUT DECAWAVE

DecaWave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4 standard UWB PHY. This device is the first in a family of parts.

The resulting silicon has a wide range of standards-based applications for both Real Time Location Systems (RTLS) and Ultra Low Power Wireless Transceivers in areas as diverse as manufacturing, healthcare, lighting, security, transport, and inventory and supply-chain management.

For further information on this or any other DecaWave product contact a sales representative as follows: -

DecaWave Ltd,
Adelaide Chambers,
Peter Street,
Dublin 8,
Ireland.

mailto:sales@decawave.com
http://www.decawave.com/