

SOURCE CODE GUIDE

DECARANGING (ARM) SOURCE CODE

Understanding and using the DW1000 DecaRanging source code

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

Copyright © 2015 Decawave Ltd

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 ARM source code and the DecaRanging 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

Copyright (c) 22/04/2015 by Decawave Limited. All rights reserved.

TABLE OF CONTENTS

1	INTRODUCTION	7
2	BUILDING AND RUNNING THE CODE	8
2.1	EXTERNAL LIBRARIES	8
2.2	BUILDING THE CODE	8
3	PROGRAMMING EVB1000	9
3.1	PURCHASE THE ST-LINK/V2 JTAG PROGRAMMER	9
3.2	INSTALL ST-LINK DRIVER UTILITY	9
3.3	CONNECT ST-LINK TO THE EVB1000 EVALUATION BOARD AND LOADING THE BUILT IMAGE	9
4	OVERVIEW	12
5	DETAILED DESCRIPTION OF DECARANGING CODE STRUCTURE	13
5.1	TARGET SPECIFIC CODE	13
5.2	ABSTRACT SPI DRIVER – SPI LEVEL CODE.....	14
5.3	DEVICE DRIVER – DW1000 DEVICE LEVEL CODE	14
5.4	INSTANCE CODE	14
5.5	TOP LEVEL APPLICATION CODE	16
5.6	FOLDER STRUCTURE	17
6	RANGING ALGORITHM	18
6.1	DECARANGING’S TAG/ANCHOR TWO-WAY RANGING ALGORITHM	18
6.2	MESSAGES USED IN DECARANGING’S TAG/ANCHOR TWO-WAY RANGING	19
6.2.1	<i>General ranging frame format</i>	19
6.2.2	<i>Blink frame format</i>	20
6.2.3	<i>Poll message</i>	21
6.2.4	<i>Response message</i>	21
6.2.5	<i>Final message</i>	22
6.2.6	<i>Ranging Initiation message</i>	22
6.3	FRAME TIME ADJUSTMENTS	23
6.3.1	<i>Frame Transmit-Time Adjustment</i>	23
6.3.2	<i>Frame Receive-Time Adjustment</i>	23
7	CODE / SYSTEM ISSUES	24
7.1	ANTENNA DELAY	24
8	OPERATIONAL FLOW OF EXECUTION	25
8.1	THE MAIN APPLICATION ENTRY	25
8.2	INSTANCE STATE MACHINE	25
8.2.1	<i>Initial state: TA_INIT</i>	25
8.2.2	<i>State: TA_TXBLINK_WAIT_SEND</i>	26
8.2.3	<i>State: TA_TXPOLL_WAIT_SEND</i>	26
8.2.4	<i>State: TA_TXE_WAIT</i>	27
8.2.5	<i>State: TA_TX_WAIT_CONF</i>	27
8.2.6	<i>State: TA_RXE_WAIT</i>	27
8.2.7	<i>State: TA_RX_WAIT_DATA</i>	27
8.2.8	<i>State: TA_SLEEP_DONE</i>	28
8.2.9	<i>State: TA_TXE_WAIT</i>	28

8.2.10	State: TA_TXFINAL_WAIT_SEND.....	28
8.2.11	State: TA_TX_WAIT_CONF (for Final message TX)	29
8.2.12	CONCLUSION	29
9	BIBLIOGRAPHY	30
10	DOCUMENT HISTORY	30
11	MAJOR CHANGES	30
11.1	RELEASE 1.7	30
11.2	RELEASE 1.8	31
11.3	RELEASE 1.9	31
11.4	RELEASE 2.0	31
11.5	RELEASE 2.1	31
12	FURTHER INFORMATION	32

1 INTRODUCTION

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

This document should be read in conjunction with the "*EVK1000 User Guide*" which gives an overview of DW1000 evaluation kit (EVK1000) and describes how to operate the DecaRanging Application.

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

Section 8 - 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:

"DecaRanging MP 3.11" application version and
"DW1000 Device Driver Version 04.00.04" 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](#)".

2 BUILDING AND RUNNING THE CODE

2.1 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 [1]

2.2 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 DecaRanging ARM software project we use the GNU Tools ARM for Embedded.

In Figure 1 the user should enter the path to ARM tools for embedded toolchain – e.g. “C:\GNUToolsARMEEmbedded\4.8_2014q1\bin”. GNU Tools ARM for Embedded can be found: <https://launchpad.net/gcc-arm-embedded>

CooCox IDE can be downloaded from: <https://www.cocox.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. A user can also select a different toolchain as shown in Figure 10.

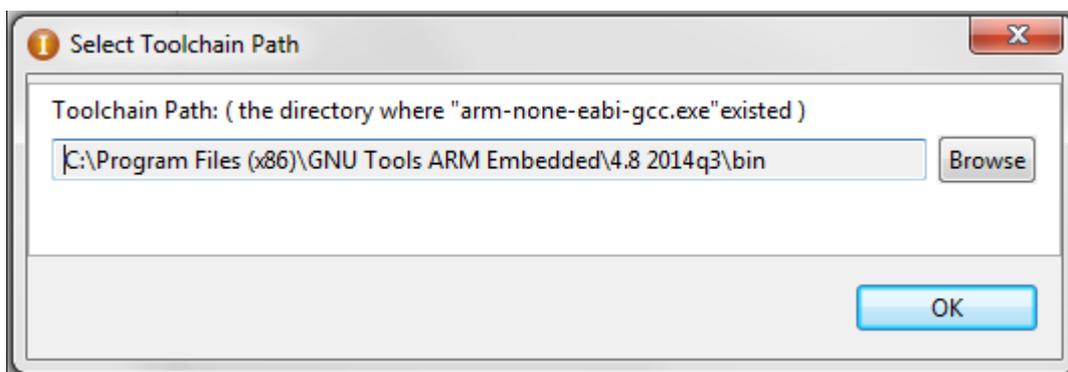


Figure 1: Select toolchain path

3 PROGRAMMING EVB1000

This chapter details how to download the DecaRanging ARM binary into the STM32 on the EVB1000 HW.

3.1 Purchase the ST-Link/V2 JTAG Programmer

Decawave uses the [ST-LINK/V2](http://www.st.com/) JTAG Programmer to connect to the EVB1000 board and program the code. This inexpensive JTAG tool is from STMicroelectronics or their distributors see (<http://www.st.com/>).



Figure 2: ST-Link/V2 JTAG Programmer

3.2 Install ST-LINK driver utility

Go to page <http://www.st.com/web/en/catalog/tools/PF258168>

This allows you to download ST Microelectronics part # STSW-LINK004 which is identified as the **STM32 ST-LINK utility**

Click on the download link to download file “stsw-link004.zip” which contains the “STM32 ST-LINK Utility_v2.4.0.exe” installer

Extract the Installer exe from inside the downloaded zip file and run it. This will install the ST-Link utility and driver software. We used default installation directories and options. Click “Next” button repeatedly and “finish” button at the end.

3.3 Connect ST-LINK to the EVB1000 Evaluation Board and loading the built image

The EVB1000 has a standard 20-PIN JTAG connector. The connections are shown in the figures below.

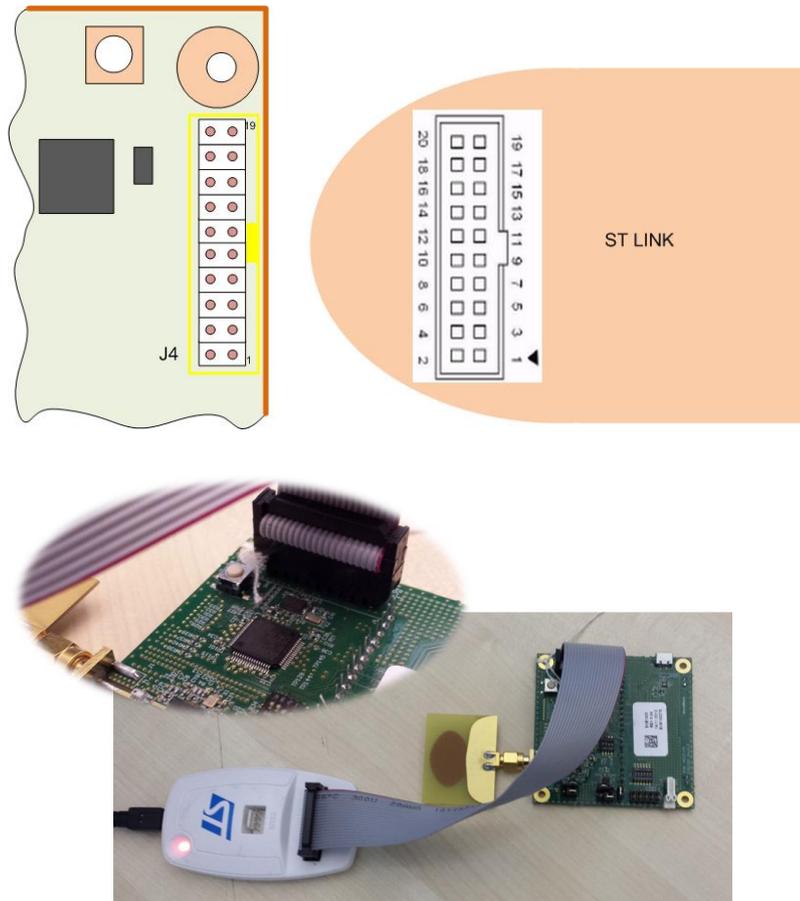


Figure 3: ST-LINK Connections

Connect ST-LINK as shown above and power up the EVB1000.

Run the ST-link “STM32 ST-LINK Utility” and in “Target” menu select “Settings” sub-option, and select SWD connection mode.

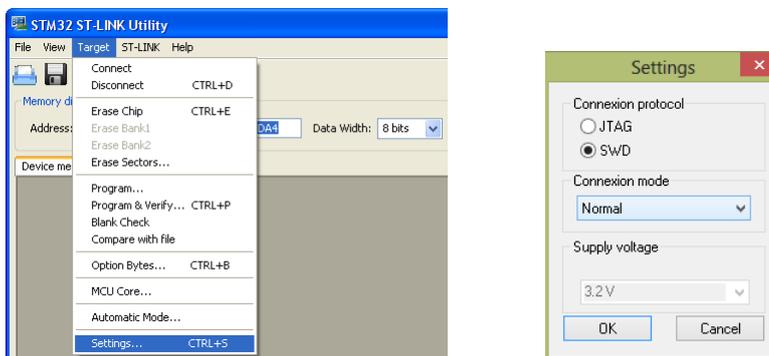


Figure 4: ST-LINK Utility Menus

- Use “Target” menu “Connect” sub-option to connect to the target device. The progress/status pane at the bottom will inform you of progress:

: Connected via SWD.

: Device ID: 0x418

: Device flash Size: 256 Kbytes

: Device family: STM32F10xxx Connectivity Line

- Use “File” menu “Open File...” sub-option to browse to and find the binary file called “DecaRanging_ARM.bin”.
- Next select “Target” menu sub-option “Program & Verify” and click start when dialog below opens.

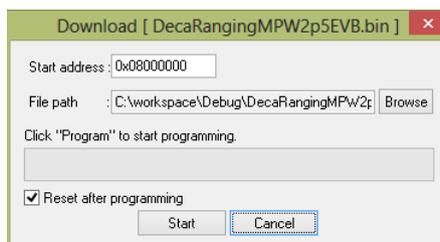


Figure 5: ST-LINK Progress

- The programming and verification will be done and the progress/status pane at the bottom will inform you of progress:
 - : [DecaRanging_ARM.bin] opened successfully.
 - : Flash memory programmed in 10s and 766ms.
 - : Verification...OK
- Once programming is complete you can disconnect the ST-Link and re-power the unit to begin execution of the newly loaded program.
- Tool bar icons  can also be used to connect and disconnect the JTAG controller.

4 OVERVIEW

Figure 6 below shows the layered structure of the DecaRanging application, giving the names of the main files associated with each layer and a brief description of the functionality provided at that layer.

<u>Name of file</u>	<u>Layer</u>	<u>Functional Description</u>
dw_main.c	Application	Runs "Instance", display results, provides interface for user control and configuration
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 6: Software layers in DecaRanging

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

5 DETAILED DESCRIPTION OF DECARANGING CODE STRUCTURE

With reference to [Figure 6](#), the identified layers are described in more detail below.

5.1 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 “[EVK1000 User Guide](#)” 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).

5.2 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.

5.3 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.

5.4 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 ranging 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.

Initially the unpaired anchor and tag are in a discovery phase where the unpaired tag sends a *Blink* message that contains its own address, after which it listens for a *Ranging Initiation* response from an anchor. If it does not get one it sleeps for a period (default of 1 second) before blinking again. The unpaired anchor listens for tag blink messages. The anchor will then pair with a first tag it gets the *Blink* message from, and send the *Ranging Initiation* message to exit from the *Discovery Phase* and enter *Ranging Phase*.

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 *DecaRanging* 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 7](#) shows the arrangement and general operation of the two-way ranging as implemented by the DecaRanging application.

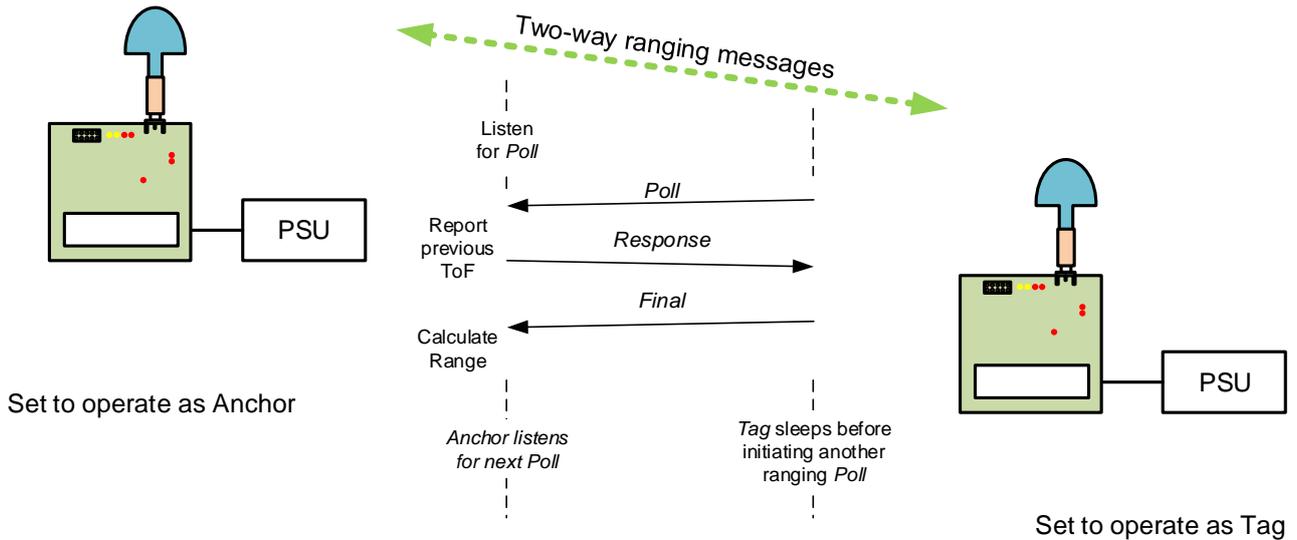


Figure 7: Two way ranging in DecaRanging

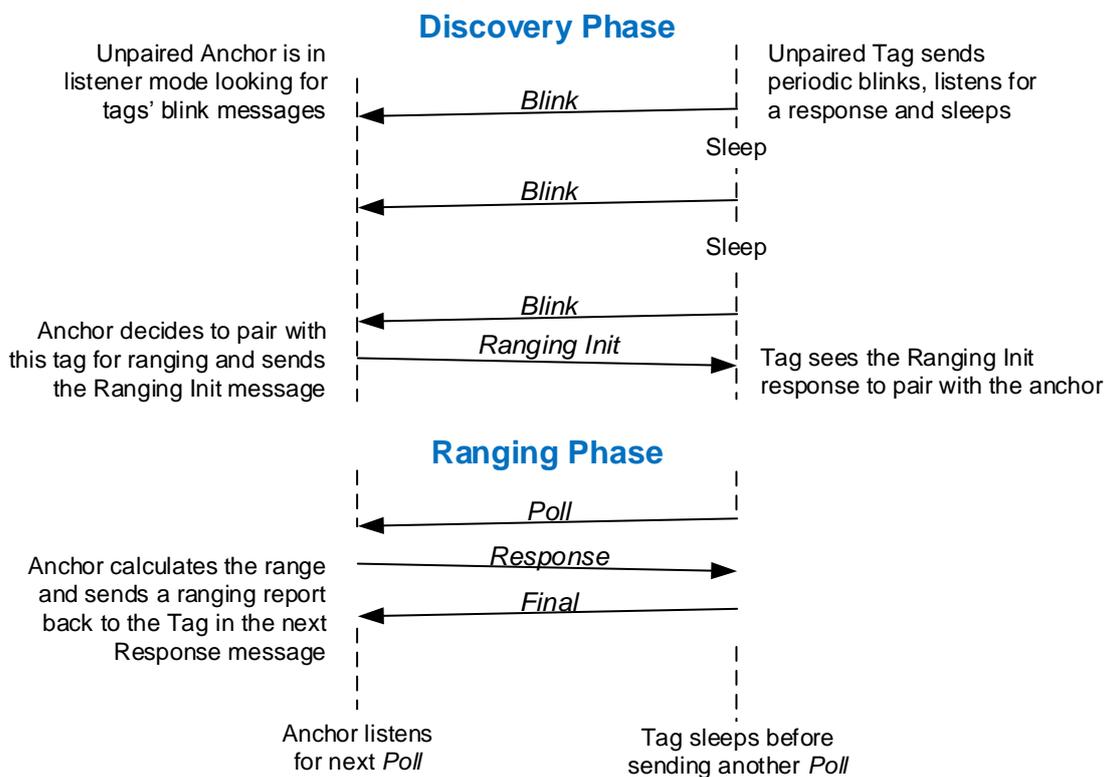


Figure 8: Discovery and Ranging phase message exchanges

Once the anchor enters the *Ranging* phase it turns on its receiver and waits indefinitely for a poll message. The tag sends a *Poll* message, and then waits for a *Response* message from the anchor, after which it sends a *Final* message. At the end of this exchange the anchor calculates the range to the tag. The anchor will include the TOF in the next response message to the tag. If the anchor response is not received the tag times out and sends the *Poll* message again (after 500 ms sleep period). Section 6 describes the ranging algorithm in more detail including the format of the messages exchanged and the calculations performed.

Above this instance level is the application that provides the user interface described in section 5.5 below. The reader is directed to Section 8 - Operational flow of execution and to the code in files [instance.c](#) and [instance_common.c](#) for more details on this instance layer.

5.5 Top level Application code

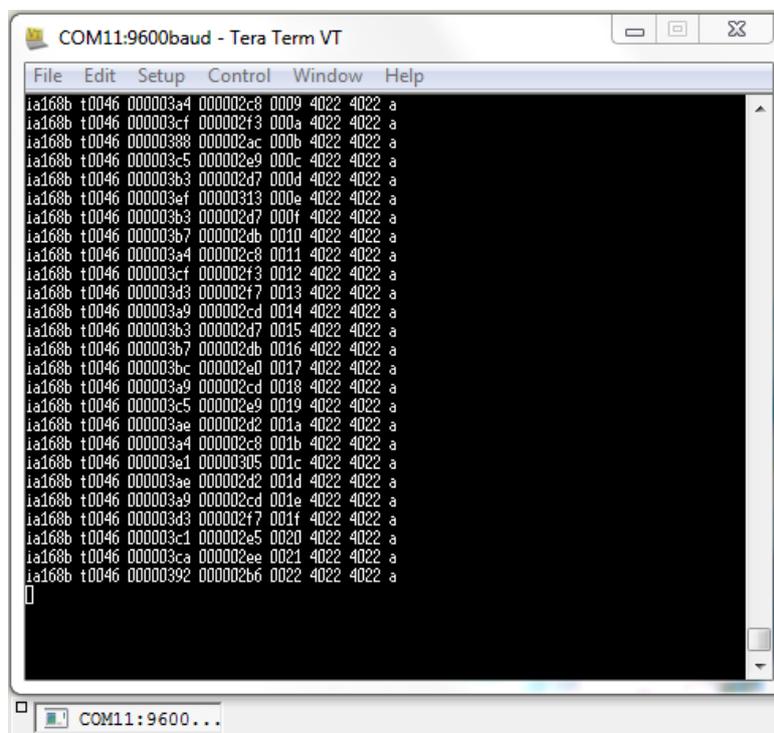
The top level application ([main.c](#)) contains the main entry point for the DecaRanging ranging demo application and also all the user interface code.

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.



Figure 9: Example LCD display showing last and average range

The application also outputs some debug information over the virtual COM port. Figure 10 shows example output from an anchor. Each TOF report starts with “i” then aXXXX and tYYYY where XXXX are the 16 LSBs of anchor address and YYYY are the 16 LSBs of tags address. The third column is the range in mm (32 bit hex number) after bias correction has been applied, the fourth column is the raw range in mm (32 bit hex number). This is followed by number of ranges (16 bit hex number) and then TX and RX antenna delays (16 bit hex number). The last column is “t” for a tag and “a” for an anchor.



```

COM11:9600baud - Tera Term VT
File Edit Setup Control Window Help
ia168b t0046 000003a4 000002c8 0009 4022 4022 a
ia168b t0046 000003cf 000002f3 000a 4022 4022 a
ia168b t0046 00000388 000002ac 000b 4022 4022 a
ia168b t0046 000003c5 000002e9 000c 4022 4022 a
ia168b t0046 000003b3 000002d7 000d 4022 4022 a
ia168b t0046 000003ef 00000313 000e 4022 4022 a
ia168b t0046 000003b3 000002d7 000f 4022 4022 a
ia168b t0046 000003b7 000002db 0010 4022 4022 a
ia168b t0046 000003a4 000002c8 0011 4022 4022 a
ia168b t0046 000003cf 000002f3 0012 4022 4022 a
ia168b t0046 000003d3 000002f7 0013 4022 4022 a
ia168b t0046 000003a9 000002cd 0014 4022 4022 a
ia168b t0046 000003b3 000002d7 0015 4022 4022 a
ia168b t0046 000003b7 000002db 0016 4022 4022 a
ia168b t0046 000003bc 000002e0 0017 4022 4022 a
ia168b t0046 000003a9 000002cd 0018 4022 4022 a
ia168b t0046 000003c5 000002e9 0019 4022 4022 a
ia168b t0046 000003ae 000002d2 001a 4022 4022 a
ia168b t0046 000003a4 000002c8 001b 4022 4022 a
ia168b t0046 000003e1 00000305 001c 4022 4022 a
ia168b t0046 000003ae 000002d2 001d 4022 4022 a
ia168b t0046 000003a9 000002cd 001e 4022 4022 a
ia168b t0046 000003d3 000002f7 001f 4022 4022 a
ia168b t0046 000003c1 000002e5 0020 4022 4022 a
ia168b t0046 000003ca 000002ee 0021 4022 4022 a
ia168b t0046 00000392 000002b6 0022 4022 4022 a

```

Figure 10: Example Teraterm Window showing the debug info sent via COM port

5.6 Folder structure

Table 1 gives the folder structure of the DecaRanging application source code given along with a brief description of each folder's content. Table 2 shows the folder structure for the ST System Workbench project. The reader is referred to the other sections of this document for more details on the code structure and organisation.

Table 1: List of folders in the DecaRanging application

Folder		Brief description
Libraries		ARM and STM32 low-level layers
	CMSIS	Hardware abstraction layer for ARM Cortex-M processors
	STM32_USB_Device_Library	USB library and device driver for STM32 processors
	STM32_USB_OTG_Driver	
	STM32F10x_StdPeriph_Driver	Hardware abstraction layer for ST STM32 F1 processors
src		DecaRanging's specific code
	application	DecaRanging's high level application and instance layers
	compiler	Standard libraries inclusions
	decadriver	DW1000 device driver
	platform	High level driver (including interrupt management) for various ST and DW peripherals
	sys	Standard libraries function reimplementation
	usb	High level USB driver

Table 2: List of folders for the ST System Workbench project

Folder		Brief description
Drivers		ARM and STM32 low-level layers
	CMSIS	Hardware abstraction layer for ARM Cortex-M processors
	STM32F1xx_HAL_Driver	HAL driver for STM32 processors
	STM32F10x_StdPeriph_Driver	STM 32 Peripheral drivers for F1xx series
Middlewares/ ST	STM32_USB_Device_Library	USB library files
Inc		Header files
Src		DecaRanging's specific code
	application	DecaRanging's high level application and instance layers
	compiler	Standard libraries inclusions
	decadriver	DW1000 device driver
	platform	High level driver (including interrupt management) for various ST and DW peripherals
	usb	High level USB driver

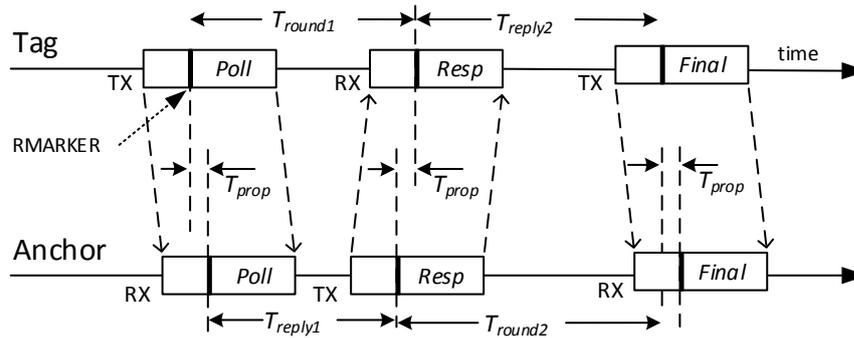
6 RANGING ALGORITHM

This section describes the ranging algorithm used in the DecaRanging ranging demo application. In contrast to some earlier versions of DecaRanging, the ranging algorithm in this code is quite efficient for two-way ranging requiring just three messages to be exchanged for an accurate range to be calculated. This is described below.

6.1 DecaRanging'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 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 time out and send the poll again.
- 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} . The anchor also includes the time-of-flight (TOF) calculated for the previous ranging exchange.
- 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 tag also gets the TOF from the previous ranging exchange and displays the distance.
- The anchor receiving this *Final* response message (at T_{RF}) now has enough information to work out the range. $T_{round1} = T_{RR} - T_{SP}$; $T_{reply1} = T_{SR} - T_{RP}$; $T_{round2} = T_{RF} - T_{SR}$; $T_{reply2} = T_{SF} - T_{RR}$.
- It is to be noted that, for small ranges, a received signal level bias correction has to be applied to calculated raw range. More details about this bias correction can be found in APS011 "Sources of error in TWR schemes".
- In the DecaRanging ranging demo the anchor will send the calculated TOF to the tag to give it something to display. This TOF will be sent in the next Response message. Figure 11 shows this exchange and gives the formula used in the calculation of the range.
- After this the anchor turns on its receiver again to await the next poll message, while the tag meanwhile sleeps or counts off the delay period to the next ranging attempt.



The *Final* message communicates the tag's T_{round} and T_{reply} times to the anchor, which calculates the range to the tag as follows:

$$T_{prop} = \frac{T_{round1} \times T_{round2} - T_{reply1} \times T_{reply2}}{T_{round1} + T_{round2} + T_{reply1} + T_{reply2}}$$

Figure 11: Range calculation in DecaRanging

6.2 Messages used in DecaRanging's tag/anchor two-way ranging

Five messages are employed in the tag/anchor two-way ranging, two in the *Discovery* phase (the blink and ranging initiation messages) and three in the *Ranging* phase (the poll message, the response message, the final message), as shown in Figure 8. Although these follow IEEE message conventions, these are NOT standard RTLS messages, the reader is referred to ISO/IEC 24730-62 (currently a draft international standard) for details of message formats being standardised for use in RTLS systems based on IEEE 802.15.4 UWB. The formats of the messages used in the demo are given below.

6.2.1 General ranging frame format

The general message format is the IEEE 802.15.4 standard encoding for a data frame. Figure 12 shows this format. The two byte Frame Control octets are constant for the DecaRanging application because it always uses data frames with 8-octet (64-bit) source and destination addresses, and a single 16-bit PAN ID (value 0xDECA). The only exception is the *Blink* message which is described in 6.2.2 below. In 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.

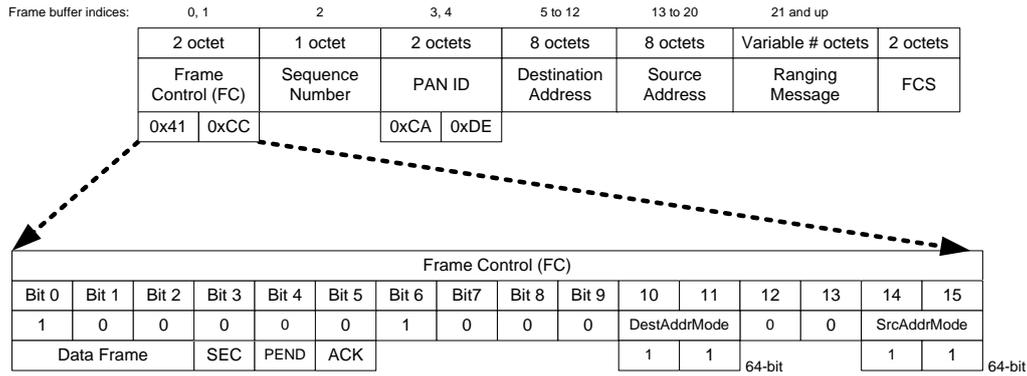


Figure 12: General ranging frame format

The sequence number octet is incremented modulo-256 for every frame sent, in line with IEEE rules.

The source and destination addresses are 64-bit numbers programmed uniquely into each device (during EVB1000 manufacture). This can be used by the application to give each DW1000 based product a unique address.

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 three ranging messages it is. These are shown in Figure 14 and described in sections 6.2.3 to 6.2.6. 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 12 to form the complete ranging message in each case.

6.2.2 Blink frame format

The special *Blink message* frame format is used for sending of the Tag Blink messages. The blink frame is simply sent without any additional application level payload, i.e. the application data field of the blink frame is zero length. The result is a 12-octet blink frame. The encoding of the minimal blink is as shown in Figure 13.

1 octet FC	1 octet	8 octets	2 octets
0xC5	Seq. Num	64-bit Tag ID	FCS

Figure 13: the 12-octet minimal blink frame

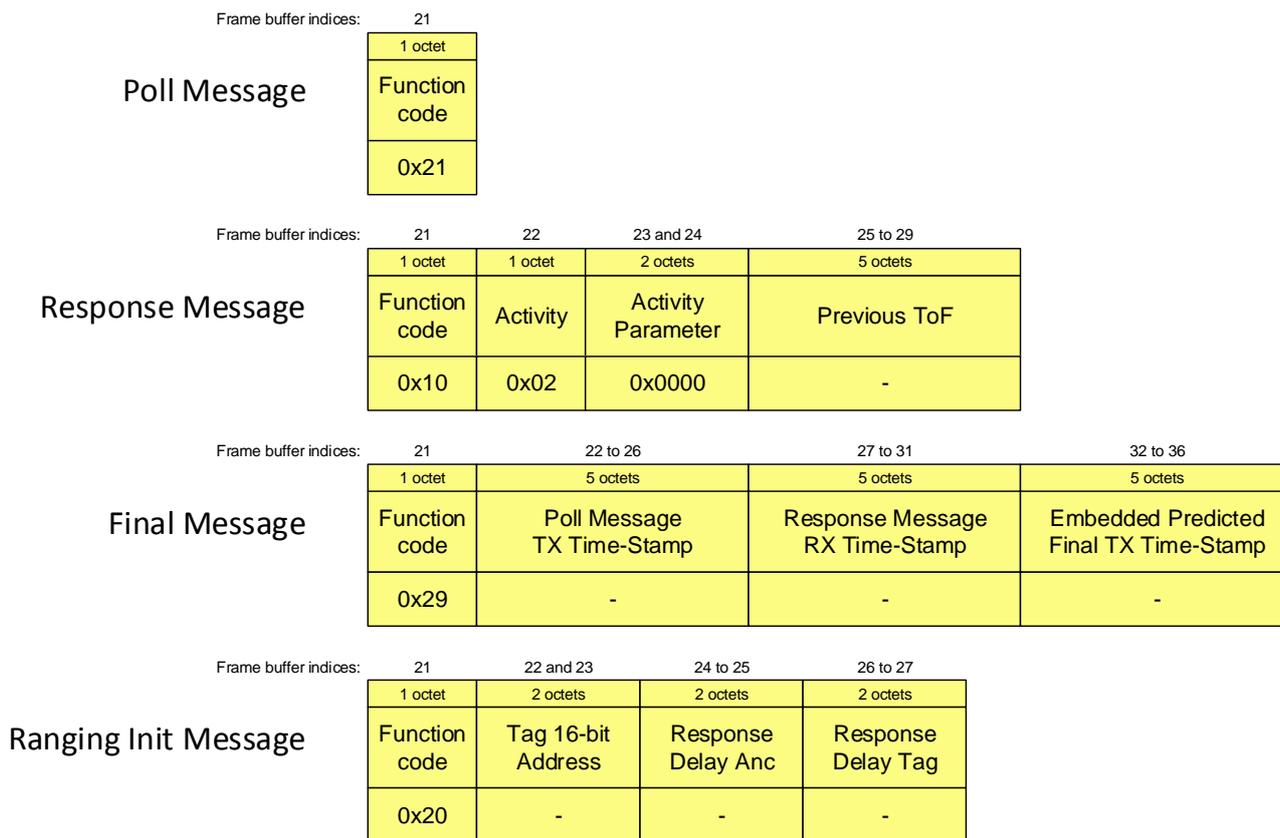


Figure 14: Ranging message encodings

6.2.3 Poll message

The poll message is sent by the tag to initiate a range measurement. For the poll message, the ranging message portion of the frame is a single octet, with value: 0x21.

6.2.4 Response message

The response message is sent by the anchor in response to a poll message from the tag. For the response message a single octet would be sufficient, but to allow for some future expansion possibilities a more complex encoding has been included. Table 3 lists and describes the individual fields within the response message.

Table 3: Fields within the ranging response message

Octet #'s	Value	Description
1	0x10	This octet 0x10 identifies this as an anchor response controlling the activity of the tag
2	0x02	This activity octet tells the tag to continue with the ranging exchange
3 to 4	0x0000	This two octet parameter is unused for activity 0x02.
5 to 9	-	This five octet field is the TOF from the previous ranging exchange. 40-bit DW1000 time-units.

6.2.5 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 4 lists and describes the individual fields within the final message.

Table 4: Fields within the ranging final message

Octet #'s	Value	Description
1	0x29	This octet identifies the message as the tag "Final" message
2 to 6	-	This five octet field is the TX timestamp for the tag's poll message, i.e. the precise time the frame was transmitted.
7 to 11	-	This five octet field is the RX timestamp for the response poll message, i.e. the time the tag received the response frame from the anchor.
12 to 16	-	This five octet field is the TX timestamp of this final message, i.e. the precise time the frame was (or will be) transmitted, this needs to be calculated by the tag as described in section 6.2.5.1 below.

6.2.5.1 Final message embedded TX timestamp

The final message includes a field that is its own transmit timestamp. The tag microprocessor needs to pre-calculate this and embed it in the message buffer before initiating the transmission of the final message. Assuming that it has already calculated DT, the reply time to programme as the delayed send time for the message, the embedded time is then just DT masked to clear the lower 9 bits, plus the TX antenna delay value.

In the DecaRanging source code this calculation is done in file [instance.c](#) in state TA_TXFINAL_WAIT_SEND.

6.2.6 Ranging Initiation message

Upon receiving the *Blink message* the unpaired anchor will send the *Ranging Initiation message* to the tag that has sent the blink message. The ranging initiation message is 5 octets in length. Table 5 lists and describes the individual fields within the ranging initiation message.

Table 5: Fields within the ranging initiation message

Octet #'s	Value	Description
1	0x20	This octet 0x20 identifies the message as a range report
2 to 3	-	This 16-bit field can be used by tag to change to use the specified 16-bit address. Instead of 64-bit address.
4 to 5	-	This 16-bit bit field gives the anchor response time to be used in the following ranging exchange: <ul style="list-style-type: none"> - bit 0 to 14: value - bit 15: 0 for microseconds, 1 for milliseconds
6 to 7	-	This 16-bit bit field gives the tag response time to be used in the following ranging exchange: <ul style="list-style-type: none"> - bit 0 to 14: value - bit 15: 0 for microseconds, 1 for milliseconds

6.3 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 EVK1000 kit are paired and the antenna delays are calibrated, and programmed into the DW1000 OTP (one-time-programmable) memory. However if a value has not been programmed the DecaRanging application will use the default antenna delay value as set in the [instance.h](#) file, there are two values, one for each PRF configuration (DWT_PRF_64M_RFDLY (515.6f); DWT_PRF_16M_RFDLY (515.0f)). 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.

The individual adjustments made to correct the timestamps are discussed below.

6.3.1 Frame Transmit-Time Adjustment

In the DW1000 the transmit time stamp is made as the RMARKER is sent by the digital circuitry. If the TX_ANTD register value is programmed it will be automatically added to the reported TX timestamp, and no software adjustment is necessary.

6.3.2 Frame Receive-Time Adjustment

In the DW1000, the receive time stamp is initially made as an appropriate event representing the receipt of the RMARKER detected digital circuitry, and then a first path seek algorithm is run to find the first path more precisely, and finally the value is adjusted by subtracting the configured RX antenna delay value. This final adjusted RX timestamp is saved in the register and the software does not have to make any further adjustments to the time of arrival read from the IC register.

7 CODE / SYSTEM ISSUES

7.1 *Antenna Delay*

The antenna delay may need changing if a different antenna is being used.

Note: If the antenna delay value is set too large, it results in negative RTD calculation results (internally to the software) and these RTD values are discarded as bad and no RTD / distance measurement will be reported. In using the system, if the communication seems to be working, but the time-of-flight status lines are not updating, then this may be because the antenna delay is set to too large a value. This can be checked by clearing the antenna delays to zero. To tune the antenna delay to the correct value is a process of trial and error, tweaking the antenna delay until the average distance reported matches the real antenna-to-antenna distance measured with a tape measure.

8 OPERATIONAL FLOW OF EXECUTION

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

8.1 *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. 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.

8.2 *Instance state machine*

The instance state machine delivers the primary DecaRanging 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.

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 can be run periodically or as a result of a pending interrupt. It checks if there are any outstanding events that need to be processed and calls the `testapprun()` function to process them. It also reads the message/event counters and checks if any timers have expired. Below paragraphs describe the `testapprun()` state machine in detail:

8.2.1 *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 or anchor 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 send a *Blink* message to allow an anchor to discover the tag and then initiate a ranging exchange, thus the state “`inst->testAppState`” is changed to “`TA_TXBLINK_WAIT_SEND`”.

8.2.2 State: `TA_TXBLINK_WAIT_SEND`

In the state “`case TA_TXBLINK_WAIT_SEND`”, we want to send the *Blink* message, so firstly we set up the message frame control data and then fill the rest of the message with the tag address. After sending the *Blink* message (using immediate send option with response expected parameter set), the state machine state will be changed to “`TA_TX_WAIT_CONF`”, where the tag awaits confirmation of the frame transmission.

As the `testapprun()` state machine state is set to “`TA_TX_WAIT_CONF`”, and as that state has more than one use, “`inst->previousState = TA_TXBLINK_WAIT_SEND`” is set to as a control variable.

Before starting the transmission we also configure the receiver turn on delay and RX frame wait timeout. Receiver turn on delay is specified by `inst->rnginitw4Rdelay_sy` and RX frame wait timeout is specified by `inst->fwtoTimeB_sy`. The delays and timeouts are calculated as part of initialisation of the application by `instancesetreplydelay()` function.

As the transmission command had `DWT_RESPONSE_EXPECTED` set the receiver will turn on automatically and then time out if no message is received. After timing out the tag will go to sleep (enter DEEP SLEEP mode) and the microprocessor will wake it up after `tagBlinkSleepTime_ms` to restart blinking (this is done in “`TA_SLEEP_DONE`” state).

8.2.3 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 `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. 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 8.2.10 below), it is required to send the message at an exact 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 `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.

¹ The “`TA_`” prefix is because these are states in the “Test Application”.

8.2.4 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) or before the next *Blink* message is sent. Here we check if tag needs to enter a sleep mode before the next *Poll* or *Blink* messages are sent, and call `dwt_entersleep()` if sleep is required.

8.2.5 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) to turn on the receiver and await a response message. The next state is thus set “`inst->testAppState = TA_RXE_WAIT`”. See 8.2.6 below for details of what this does.

8.2.6 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.

8.2.7 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 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 14, for details.

For the point of view of the discussions here the tag is awaiting the anchor’s response or ranging initiation message so we would expect the FCODE to match “RTLS_DEMO_MSG_ANCH_RESP” or “RTLS_DEMO_MSG_RNG_INIT” when in Discovery phase. 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
```

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 interrupt². When the timeout happens the tag will go back to restart the ranging exchange.

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

8.2.8 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;`

Note: In order to minimise power, the microprocessor uses DW1000 RSTn pin to notify when the DW1000 enters the IDLE 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 IDLE state.

8.2.9 State: TA_TXE_WAIT

In this state “`case TA_TXE_WAIT`”, the tag checks if it needs to go to sleep (low power state) before starting a new ranging exchange. If it comes into this state from sleep it will proceed to send the next *Poll* or *Blink* message.

Note: To save power the tag could poll one anchor and then “sleep” for a length of time before polling the same anchor again. In the TOF RTLS system, a tag might poll and range with a number of anchors and then enter a sleep mode before starting the process again.

8.2.10 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.

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

In the state “`case TA_TX_WAIT_CONF`”, (as detailed in section 8.2.5 above) we await the confirmation that the message transmission has completed.

When we get this, we use the “`inst->previousState == TA_TXFINAL_WAIT_SEND`” to identify that we are a tag who has just sent the final and we go on to send another poll message (perhaps after a sleep period of inactivity).

8.2.12 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 blink message. Then it will associate with the tag that sent it and send the ranging initiation 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.

9 BIBLIOGRAPHY

Ref	Author	Title
[1]	Decawave/ST	01_Installation of the tools and drivers.pdf
[2]	Decawave	DW1000 Data Sheet
[3]	Decawave	DW1000 User Manual
[4]	Decawave	EVK1000 User Manual
[5]	Decawave	DecaRanging Ranging Demo Application (PC Version) User Guide
[6]	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/
[7]	Decawave	APS011: Sources of error in TWR schemes

10 DOCUMENT HISTORY

Revision	Date	Description
1.5	20 th December 2013	Initial release for production device.
1.7	11 th November, 2014	Scheduled update
1.8		
1.9	30 th September, 2015	Scheduled update
2.0	1 st September, 2016	Scheduled update
2.1	30 th July, 2018	Added reference to CubeMX project and updated with new Logo

11 MAJOR CHANGES

11.1 Release 1.7

Page	Change Description
All	Update of version number to 1.7
All	Various typographical changes
3	Added Decawave’s disclaimer

11.2 Release 1.8

Page	Change Description
All	Update of version number to 1.8
All	Various typographical changes
3	New Disclaimer as new source includes ST's library files
12	Updated the description of text output over VCOM port
18	Corrected Figure 9
24	Added a new tool chain for the build

11.3 Release 1.9

Page	Change Description
All	Update of version number to 1.9
All	Various typographical changes
8-9	Added sections 1 and 2
12	Updated the description of text output over VCOM port
18	Section 5 – description of the new asymmetric TWR algorithm
26	Section 7 – updated the section to reflect the new algorithm

11.4 Release 2.0

Page	Change Description
All	Update of version number to 2.0
20	Fix broken reference link.
22	Fix Table 4

11.5 Release 2.1

Page	Change Description
All	Update of version number to 2.1
All	Add references to ST System Workbench IDE project files, and new HAL generated by Cube MX tool.
All	New Logo

12 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.