TTK1000 DWM1001 TAG SOFTWARE CODE GUIDE

Understanding and using the TTK1000 DMW1001 TDoA TAG source code

Version 1.0

This document is subject to change without notice.

1 Table of Contents

List of Figures

List of Tables

2 INTRODUCTION

This document, "*TTK1000 DWM1001 Tag Software Guide*" is a guide to the application source code of the Decawave's "dwm1001_tdoa_tag" TDOA Tag application, that is a part of the TTK1000 software suit which provides the TDOA based RTLS. The Tag software is designed to run on the DWM1001 module, equipped by nRF52832 CORTEX-M4F ARM MCU.

This document should be read in conjunction with the "*DW1000 Software API Guide*" which describes the low level driver code for the DW1000 UWB transceiver IC.

The document discusses the tag source code, covering the structure of the software and the operation of the TDOA tag blinking application.

The Operational flow of execution, which is written in the style of a walkthrough of execution flow of the software, should give a good understanding of the basic operational steps of transmission, which in turn should help of integrating/porting the blinking application to a different platforms.

This document relates to the following versions:

"dwm1001_tdoa_tag" application, and
"DW1000 Device Driver Version 04.02.02" driver version

The device driver version information may be found in source code file "deca_version.h".

3 OVERVIEW

The ***Figure 1*** below shows the layered structure of the tag software, giving the names of the main files associated with each layer and a brief description of the functionality provided at that layer. The layers, functions and files involved are described in more details in the following section.

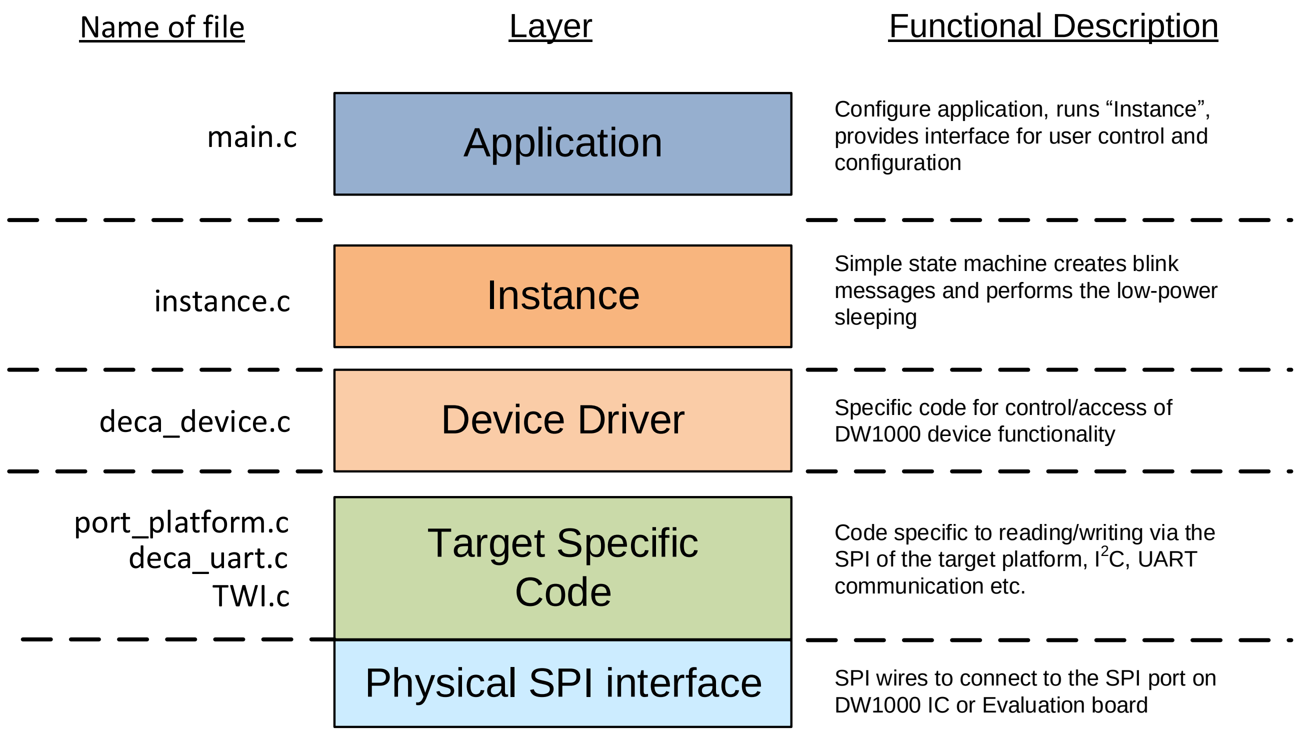

Name of file
Layer
Functional Description
main.c
Application
Configure application, runs "Instance", provides interface for user control and configuration
instance.c
Instance
Simple state machine creates blink messages and performs the low-power sleeping
deca_device.c
Device Driver
Specific code for control/access of DW1000 device functionality
port_platform.c
deca_uart.c
TWI.c
Target Specific Code
Code specific to reading/writing via the SPI of the target platform, I2C, UART communication etc.
Physical SPI interface
SPI wires to connect to the SPI port on DW1000 IC or Evaluation board

Figure 1: Software layers of the typical tag blinking application

4 BUILDING AND RUNNING THE CODE

This project is created using the Segger Embedded Studio IDE. The Segger IDE includes a full license to develop, debug and run applications targeting nRF52832 ARM MCU, which is the MCU used onto the Decawave's DWM1001 hardware platform.

4.1 INSTALLING OF THE SEGGER IDE

Download and install the Segger Embedded Studio IDE from the following web site:
https://www.segger.com/products/development-tools/embedded-studio/

4.2 LOADING OF THE PROJECT TO THE IDE

Unpack the source code to the "dwm1001_tdoa_tag" folder. In the Segger IDE, choose File->Open Solution and select the project tdoa_tag.emProject, located in the examples\tdoa_tag\ folder.

4.3 CONNECTING, BUILDING AND RUNNING OF THE APPLICATION

Connect the board to the PC. You may require to install J-Link drivers, which could be found on the Segger web site:

https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack

Install J-Link Software and Documentation pack, which includes drivers and J-Flash Lite tool needed for reprogramming new FW binary into tag.

To check if the target board is working select Target->Connect J-Link from the menu. To start the TDOA Tag application select Build->Build and Run from menu.

4.4 GEt A LICENSE FOR NORDIC MCU

If it was the first time you start your app you shall be asking for the licence. License is free for Nordic MCU'S, please refer to the Appendix A – Installing a license for Nordic MCU for details.

5 DESCRIPTION OF THE "DW_TDOA_TAG" SOURCE CODE

With the reference to the ***Figure 1*** above, the layers, corresponded to the source code of "dw_tdoa_tag" application, can be divided into following major blocks:

- Nordic BSP and drivers
- Decawave's DW1000 UWB chip device driver
- Application's target specific code
- Application itself

5.1 NORDIC DRIVERS AND LIBRARY SOURCES

The Tag hardware **DWM1001** is based on the Nordic Semiconductor nRF52832-CIAA ARM Cortex™ M4F MCU. The Board Support Package for the TDoA Tag application is shipped within the project. See the "$Project_Root/components", "$Project_Root/external" and "$Project_Root/examples/bsp".

5.2 DECAWAVE'S DW1000 DEVICE DRIVER

Decawave's DW1000 portable device driver can be found in the "$Project_Root/deca_driver" directory. It provides the interface to a library of API functions to configure and control the DW1000 UWB transceiver chip. The API functions are described in the document "*DW1000 Device Driver Application Programming Interface (API) Guide*", see the Appendix B – Bibliography.

5.3 Target Specific Code

The TDoA Tag application code (without BSP) is a platform independent and could be ported to a different platform (MCU). However, there are few target-specific dependences, where attention should be taken when porting the code to a different platform.

This code can be found in the "$Project_Root/deca_driver/port" folder. To port the project to another platform, the developer would need to replace the BSP/HAL sources with a target-specific ones, as well as rewrite the target specific source files mentioned in the above.

Below is a brief description of each file in the directory.

5.3.1 Target specific: "port_platform.c" and "deca_uart.c"

These files provides the implementation of application specific functionality to support the target hardware. This includes the Tick timer, GPIO, UART and SPI platform-dependant implementation.

Table 1 Exported functions of port_platform.c and deca_uart.c

Exported function	Description
uint32 *portGetTickCount* (**void**)	Returns current tick count, in system clock tick time (mS)
void *port_wakeup_dw1000* (**void**)	Wakes up DW1000 chip using GPIO pins
void port_set_dw1000_slowrate (***void***)	Configure SPI to work on a low speed: 2 Mbit/s

void port_set_dw1000_fastrate (void)	Configure SPI to work on a fast speed: 8 Mbit/s
void reset_DW1000 (void)	Toggle the reset pin of DW1000
void peripherals_init (void)	Initialize the peripheral modules
void deca_uart_init (void)	Configuration of the UART module
void deca_uart_receive (void)	Receive data from UART, stops after CR
void deca_uart_transmit(char *buffer)	Transmit buffer through UART
port_set_app_wakeup_check_hook (**hook_fn**)	Set up a hook function to check periodically if we need to wakeup from LP mode
void ***low_power*** (**uint32 ticks**)	Switch to LP crystal, waits for ticks msec. Any configured interrupt can interrupt low_power()
int ***readfromspi(…)***	DW1000 SPI read blocking function
int ***writetospi*(…)**	DW1000 SPI write blocking function

5.3.2 Target specific: "TWI.c"

The TWI.c file contains the target-specific code to provide the I2C I/O functionality. This required to access to an IMU sensor, which is equipped on the dwm1001 module.

Exported function	Description
void vTWI_Init(void)	Initialize enable I2C
void vTWI_Write(u8addr, u8data)	Write one byte to addr
void vTWI_Read(u8addr, u8data)	Read one byte from addr

5.4 The Application code and UART configuration

The *main.c* contains the C-entry point for the Tag application, which has a "super loop" design. This contains basic hardware and software initializations routings and serves a blinking and/or PC interface if UART is attached.

```
while (1)
{
    vTestModeMotionDetect();
    if (nrf_uart_event_check(NRF_UART0, NRF_UART_EVENT_RXDRDY))
    {
        process_uartmsg();  /* process UART msg based on user input. */
    }
    if (app.blinkenable)
    {
        instance_run();    /* transmit packet if allowed */
    }
}
```

5.4.1 UART processing

After the input string with command is received from the UART ending with the CR char (*'\r'*), *process_uartmsg()* is executed, which parses the input string and executes an appropriate command.

5.4.2 Connecting Tag to a PC

Connect the Tag to a PC and to a terminal based PC application with the following parameters: baud rate 115200, 8 data bits, no parity, 1 stop bit.

The command mode of operation is simple and is for updating of the Tag's run-time configuration. Command mode parser's and handler's functions are located in the *cmd_console.c* source file. It provides a command-line interface to configure Tag via USB-COM port. All the available commands are listed in the ***Table 2*** below.

The command line interface commands have a structure "COMMAND VALUE".

The first descriptor of the command line is the method from the class of the commands COMMAND and should be from the table below. They are not case-sensitive. The Second keyword of the command line is the VALUE , which is optional for some of the commands. Asterisk (*) indicates the default value.

Table 2 Tag Console Commands

Command	Description	
STAT	Prints out the current configuration	
HELP	Prints list of the available commands	
SAVE	This command will store the current set of the run-time variables into a Non Volatile Memory (NVM). This retain configuration even at a loss of a power source.	
CHAN	1, 2*, 3, 4, 5, 7	RF channel number
PRF	16 , 64*	Pulse Repetition Frequency 16MHz or 64MHz
PLEN	64, 128, 256, 512, 1024, 1536, 2048	Preamble length
PAC	8*, 16, 32, 64	Acquisition Chunk Size (Relates to RX preamble length)
TXCODE		TX preamble code
NSSFD		Should we use non-standard SFD for better performance or not
DATARATE	110, 850, 6810*	UWB Data Rate transfer speed 110 Kbit/s, 850 Kbit/s or 6,81Mbit/s
PHRMODE	0*, 1	PHR mode 0 for standard PHR mode and 1 for non-standard PHR mode

SFDTO	4161* / Not used	SFD detect timeout value (in symbols). This timeout used in Rx mode only, which is not affected on a TDoA Tag functionality.
SMARTPOWEREN	1*, 0	Smart Power enable / disable 1 for enable automatic Smart Tx power and 0 to disable Smart Tx power.
BLINKFAST	100-65535	Delay between blinks in ms for the moving Tag. The fastest blinking rate is 100, which means 10 blinks per second.
BLINKSLOW	100-65535	Delay between blinks in ms for the stationary Tag. The fastest blinking rate is 100, which means 10 blinks per second.
RANDOMNESS	1-50, Default 10	Randomness in % of Blinkrate time. The pause between blinks (blinkrate) would be randomized according to the value.
TAGID	-	Individual configurable ID of the Tag, 8 hex notation's chars, e.g. 123AB67E
TAGIDSET	0*, 1	Individual configurable ID of the Tag set or unset If set, then use TAGID field as a Tag address ID. If unset, then use DW1000 unique ID as a Tag address

5.5 Application: The Instance Code

The TDoA tag application has a simple main loop organisation. The instance code, located in the instance.c provides the tag blink sending functionality, while cmd_console.c implements a interface communication to a PC terminal. The *instance_run()* has to be periodically called from a main super loop, see 5.4.

The tag application is implemented by the state machine in function *testapprun()*, called from the function *instance_run()*, which is the main entry point for running of the non-blocking process. In addition to the *instance_run()* function, the instance code provides the control functions to the application. These functions are listed below to give the reader a quick review of the functionality.

Table 3 Exported functions of instance.c

Function	Description
instance_init()	Initialise instance structure.
instancereaddeviceid()	Return the Device ID register value, enables higher level validation of physical device presence.

instance_config()	Allow application configuration be passed into instance and affect underlying device operation.

The ***Figure 2*** below describes the operation flow of the *instance_run()* and its interconnections with the *testapp_run()* and *lowpower()* modules.

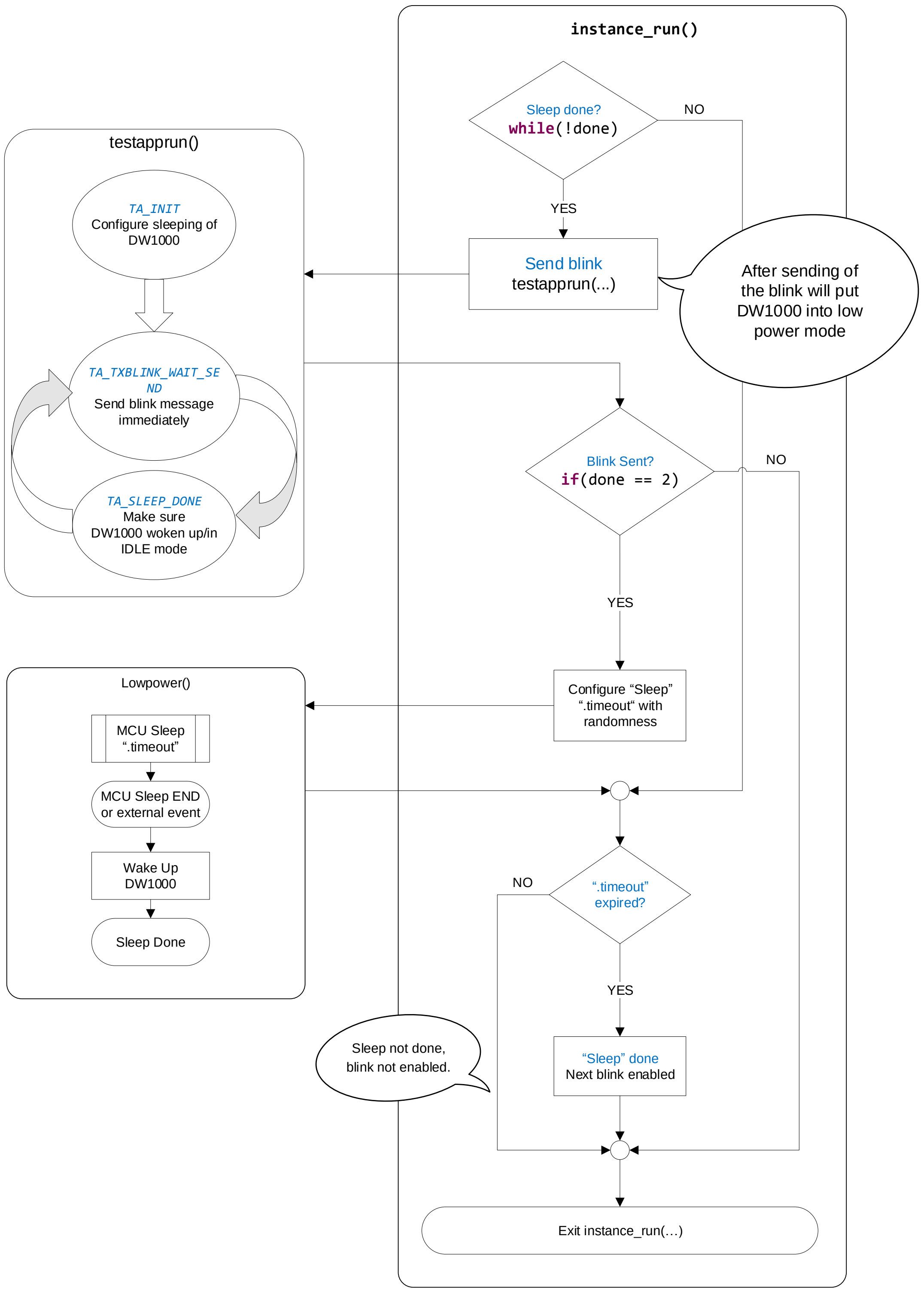
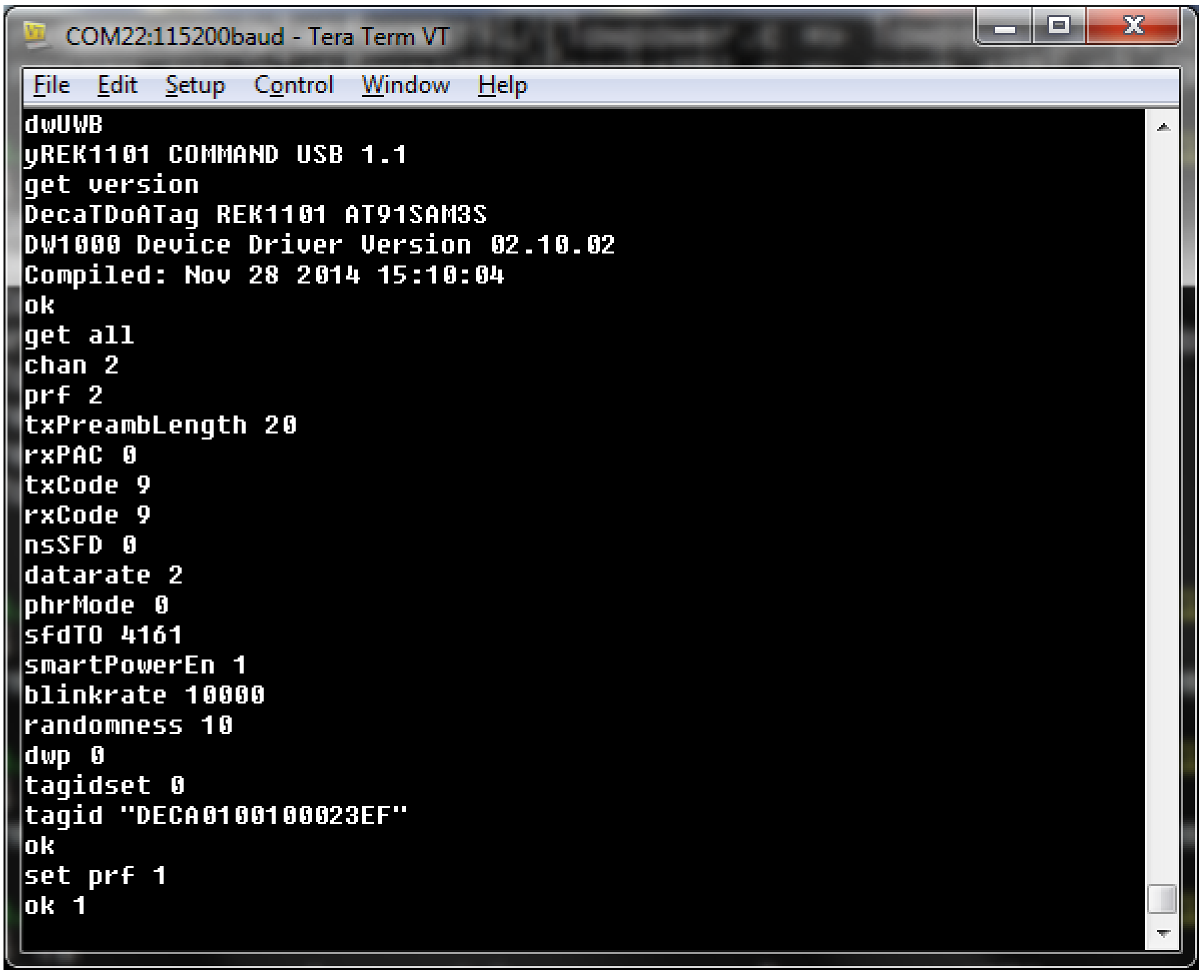


Figure 2: Overview of the Tag Blinking Instance operation


COM22:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
dwUWB
yREK1101 COMMAND USB 1.1
get version
DecaTDoATag REK1101 AT91SAM3S
DW1000 Device Driver Version 02.10.02
Compiled: Nov 28 2014 15:10:04
ok
get all
chan 2
prf 2
txPreambLength 20
rxPAC 0
txCode 9
rxCode 9
nsSFD 0
datarate 2
phrMode 0
sfdTO 4161
smartPowerEn 1
blinkrate 10000
randomness 10
dwp 0
tagidset 0
tagid "DECA0100100023EF"
ok
set prf 1
ok 1
```

Figure 3: Example Terminal window showing the TAG virtual COM port operation

5.6 Complete project's files list

The complete list of the files of the "tdoa_tag" application is in the Table 4 below. The file name is given along with a brief description of the file and its objective. Library source code from Nordic (nRF SDK) is not included in this list but is included in the project. These files are provided by Nordic Semiconductor as a part of nRF52832 SDK BSP.

Table 4: List of source files in the "dw_tag" application[1]

Filename	Brief description
deca_version.h	Decawave's version number for the DW1000 driver/API code
main.c	Application – source code (main line)
deca_device.c	Device level Functions – 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_regs.h	Device level – header (Device Register Definitions)
deca_types.h	Data type definitions – header
instance.h	Blinking Application Instance – header
instance.c	Blinking Application Instance – source code
cmd.c	Contains command line interface parser functions
cmd_fn.c	Contains command line interface functions
cmd.h	Command line interface - header
cmd_fn.h	Command line interface functions - header
default_config.h	Default tag application configuration
config.c	NVM configuration management
config.h	NVM configuration management header
pckt_ieee.h	Definition of RF messages structure
custom_board.h	Definitions for DWM1001 module (MCU pins etc)
tdoa_tag_main.c	UART communication and Acc motion detection
deca_uart.c	Low level UART communication and error handling
sdk_config.h	Definitions to enable peripherals used by SDK and main code
port_platform.c	Target-dependent functions
deca_uart.c	HW specific definitions and functions for UART Interface
deca_uart.h	Definitions for deca_uart.c
LIS2DH12.c	Middle-level communications with LIS2DH MEMS Acc via I2C
translate.c	Conversions between human-readable DW1000 settings and register values (used by serial command functions)
translate.h	Definitions for translate.c
TWI.c	Low level Two Wire Interface (I2C) for Acc communication

[1] The list of files and the names of sources could be slightly different and is subject to change without notice. Doesn't include sources from Nordic SDK

5.7 TDoA Tag Blink message format

Only one UWB transmit packet format is employed in the Tag software, as shown in **Figure 5: Blink frame byte format**

. Although these follow IEEE message conventions, this is <u>not</u> a 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 format of the packet, used in the Tag blinking demo is given below.

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 below.

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

Figure 4: the IEEE 12-octet minimal blink frame

Byte	0	1	2	3	4	5	6	7	8	9	10	11
Parameter	FC	Seq#	Tag Address								CRC	

Figure 5: Blink frame byte format

Table 5: Field definitions within the Bink frame

Parameter	Value	Description
FC	0xC5	1 byte frame control as per **Figure *4*: the IEEE 12-octet minimal blink frame**
Seq#	-	This is a modulo 256 frame sequence number.
Tag Address	-	8 byte address of the master anchor (64-bit unique address)
CRC	-	This is the frame check sequence.

6 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 SEGGER Embedded Studio) 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 it initialises the HW and various ARM MCU peripherals in the *peripherals_init()* and *TWI_Init()* functions. Then system timers starts running after *peripherals_init ().* After that, the *init_nvm_appdata()* loads from the NVM (Non-Volatile-Memory) a run-time configuration parameters (channel, PRF, data rate etc.), which are used to set up the DW1000 chip by calling the *inittestapplication()* function. Finally, the main super loop starts to execute, where the *instance_run(), the vTestModeMotionDetect()* and the *nrf_uart_event_check() functions* are called periodically to run the instance state machine and motion detection driver, as described below.

In the current TDoA Tag implementation, the DW1000 interrupt line is not required and is not serviced.

6.2 INSTANCE STATE MACHINE

The instance state machine delivers the primary TDoA blinking functionality. The *Figure 2* demonstrates chains between different parts of the program and helps in understanding of the project code operating.

The *instance_run()* function is the main function for the instance; it should be run periodically. 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. The paragraphs below describe the *testapprun()* sate machine in detail.

6.2.1 Initial state: TA_INIT

The function *testapprun()* contains the state machine that implements the blink functionality, the part of the code executed depends on the state and is selected by the “inst->testAppState” statement at the start of the function. The initial case “TA_INIT” performs initialisation and determines the next state to run. In the case of the TDoA tag the next stage is to send a *Blink* message to allow the anchors in TDoA system to calculate Tag’s timestamp and determine its position, thus the state “inst->testAppState” is changed to “*TA_TXBLINK_WAIT_SEND*”.

6.2.2 State: TA_TXBLINK_WAIT_SEND

In the state "*TA_TXBLINK_WAIT_SEND*", the *Blink* message should be send, so firstly the message frame control data has to be with the rest of the message with the tag address according to the ***Figure 5***. After setting up the DW1000 chip to transmit the *Blink* message, (using immediate send option with no response expected parameter set), the state machine is changed to "*TA_SLEEP_DONE".*

6.2.3 State: TA_SLEEP_DONE

While blink is transmitting over DW1000 transmitter, the MCU is already in in the "*TA_SLEEP_DONE*" state, and tag can go to sleep. The MCU will be waked up after tagBlinkSleepTime_ms to restart blinking.
Once the sleep timeout expires, the microprocessor will wake up the DW1000 from SLEEP. In current implementation, the registers of DW1000 are preserved, so DW1000 does not need to be configured again.

The state machine is now back to the "*TA_TXBLINK_WAIT_SEND*" and is ready to send the next blink message.

Note: Due to the nature of TDoA RTLS system, tags don't need to range to any anchors as they would in a ToF ranging system, so blinking-only tags will provide the longest battery life.

6.3 CONCLUSION

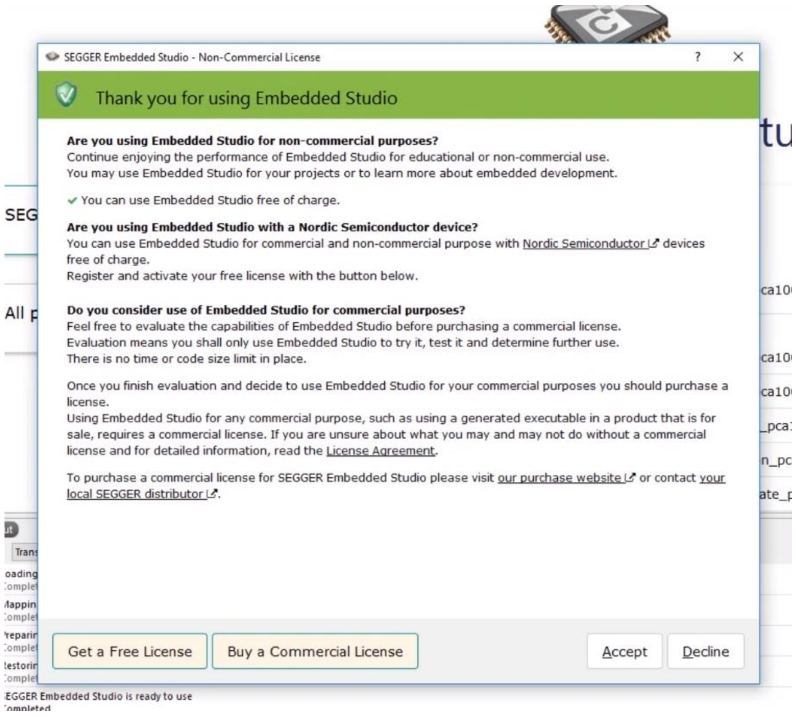
The above state descriptions should provide sufficient information on the tag state machine so that the reader can walk through the TDoA tag code.

In summary, the Tag transmits the blink message and then waits until a timer expires and then proceeds to send the next blink message.

7 APPENDIX A – INSTALLING A LICENSE FOR NORDIC MCU

At first compilation of the TDOA Tag application, the Segger IDE will require a correct license to be installed to the system.

Follow below figures to obtain a Free License for Nordic MCUs, **Figure 6**, **Figure 7**, and **Figure 8**.



SEGGER Embedded Studio - Non-Commercial License

Thank you for using Embedded Studio

Are you using Embedded Studio for non-commercial purposes?
Continue enjoying the performance of Embedded Studio for educational or non-commercial use.
You may use Embedded Studio for your projects or to learn more about embedded development.

✓ You can use Embedded Studio free of charge.

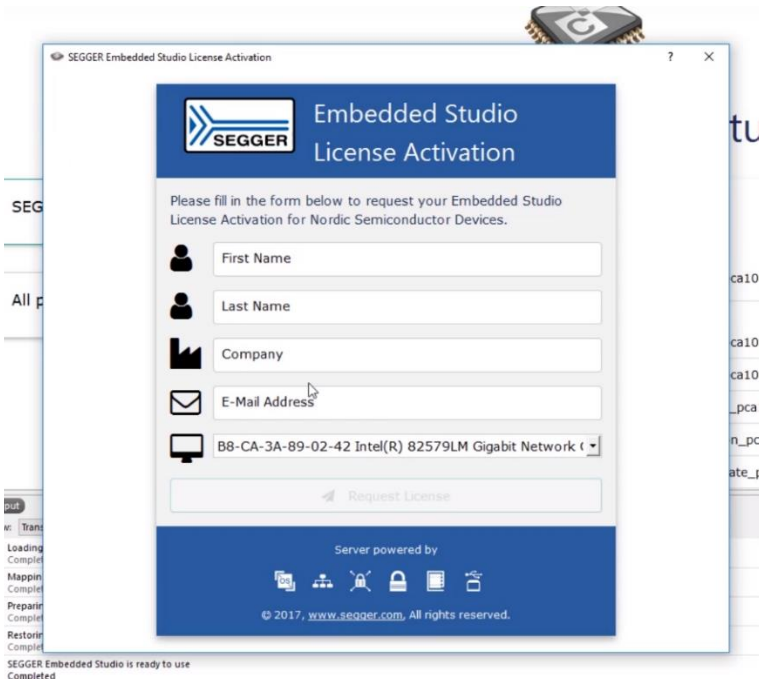
Are you using Embedded Studio with a Nordic Semiconductor device?
You can use Embedded Studio for commercial and non-commercial purpose with Nordic Semiconductor devices free of charge.
Register and activate your free license with the button below.

Do you consider use of Embedded Studio for commercial purposes?
Feel free to evaluate the capabilities of Embedded Studio before purchasing a commercial license.
Evaluation means you shall only use Embedded Studio to try it, test it and determine further use.
There is no time or code size limit in place.

Once you finish evaluation and decide to use Embedded Studio for your commercial purposes you should purchase a license.
Using Embedded Studio for any commercial purpose, such as using a generated executable in a product that is for sale, requires a commercial license. If you are unsure about what you may and may not do without a commercial license and for detailed information, read the License Agreement.

To purchase a commercial license for SEGGER Embedded Studio please visit our purchase website or contact your local SEGGER distributor.

Figure 6: Press Get a Free License



SEGGER Embedded Studio License Activation

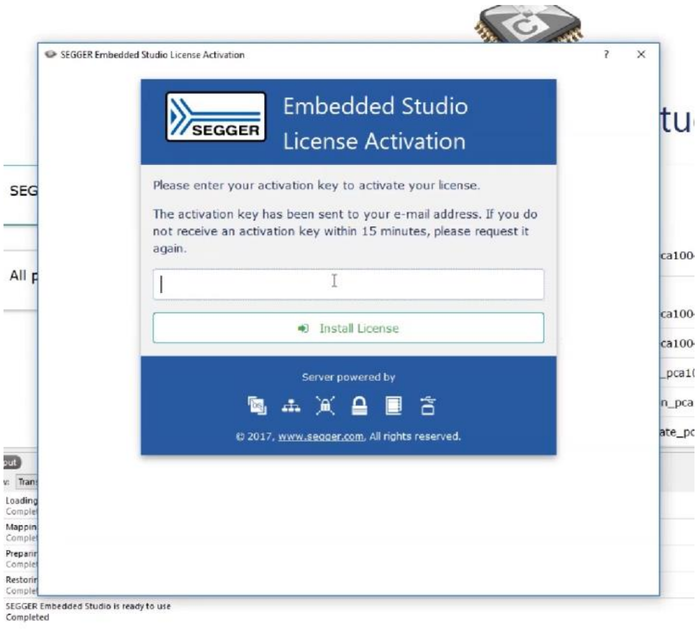
Embedded Studio License Activation

Please fill in the form below to request your Embedded Studio License Activation for Nordic Semiconductor Devices.

B8-CA-3A-89-02-42 Intel(R) 82579LM Gigabit Network (

© 2017, www.segger.com, All rights reserved.

Figure 7: Fill in your details


SEGGER Embedded Studio License Activation

Embedded Studio License Activation

Please enter your activation key to activate your license.

The activation key has been sent to your e-mail address. If you do not receive an activation key within 15 minutes, please request it again.

SEGGER Embedded Studio is ready to use
Completed

Figure 8: Enter the Activation Key you've got

For further information about the license management consult the Segger ES IDE Help and Documentation.

8 APPENDIX B – BIBLIOGRAPHY

Table 6: Table of References

1	Decawave DW1000 Datasheet
2	Decawave DW1000 User Manual
3	DW1000 Device Driver Application Programming Interface (API) Guide
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/
5	ISO/IEC 24730-62

9 DOCUMENT HISTORY

Table 7 Document History

Revision	Date	Description
1.00	6-SEP-2018	TTK1000 First release

10 FURTHER INFORMATION

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

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