

DWM1004C: TDoA TAG SOFTWARE GUIDE

Version 1.01

**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 © 2018 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

1 TABLE OF CONTENTS

| | | |
|----------|---|-----------|
| 2 | INTRODUCTION..... | 5 |
| 3 | OVERVIEW | 6 |
| 4 | BUILDING AND RUNNING THE CODE..... | 7 |
| 4.1 | LOADING OF THE PROJECT TO THE IDE..... | 7 |
| 4.2 | CONNECTING, BUILDING AND RUNNING OF THE APPLICATION | 7 |
| 5 | DESCRIPTION OF THE “DW_TDOA_TAG” SOURCE CODE..... | 9 |
| 5.1 | STM DRIVERS AND LIBRARY SOURCES..... | 9 |
| 5.2 | DECAWAVE’S DW1000 DEVICE DRIVER..... | 9 |
| 5.3 | TARGET SPECIFIC CODE..... | 9 |
| 5.3.1 | <i>Target specific: “port_platform.c”, “deca_sleep.c”, “deca_spi.c” and “deca_uart.c”</i> | <i>9</i> |
| 5.4 | THE APPLICATION CODE AND UART CONFIGURATION..... | 10 |
| 5.4.1 | <i>UART processing.....</i> | <i>11</i> |
| 5.4.2 | <i>Connecting Tag to a PC.....</i> | <i>11</i> |
| 5.5 | APPLICATION: THE INSTANCE CODE..... | 12 |
| 5.6 | COMPLETE PROJECT’S FILES LIST..... | 15 |
| 5.7 | TDOA TAG BLINK MESSAGE FORMAT | 16 |
| 6 | OPERATIONAL FLOW OF EXECUTION..... | 17 |
| 6.1 | THE MAIN APPLICATION ENTRY..... | 17 |
| 6.2 | INSTANCE STATE MACHINE..... | 17 |
| 6.2.1 | <i>Initial state: TA_INIT.....</i> | <i>17</i> |
| 6.2.2 | <i>State: TA_TXBLINK_WAIT_SEND</i> | <i>18</i> |
| 6.2.3 | <i>State: TA_SLEEP_DONE.....</i> | <i>18</i> |
| 6.3 | CONCLUSION..... | 18 |
| 7 | APPENDIX A – BIBLIOGRAPHY | 19 |
| 8 | DOCUMENT HISTORY | 20 |
| 9 | FURTHER INFORMATION | 21 |

List of Figures

| | |
|--|----|
| Figure 1: Software layers of the typical tag blinking application | 6 |
| Figure 2: Debug configuration example – Main Page | 7 |
| Figure 3: Debug configuration example – Debugger Page | 8 |
| Figure 4: Overview of the Tag Blinking Instance operation | 13 |
| Figure 5: Example Terminal window showing the TAG virtual COM port operation | 14 |
| Figure 6: the IEEE 12-octet minimal blink frame | 16 |
| Figure 7: Blink frame byte format | 16 |

List of Tables

| | |
|---|----|
| Table 1 Some target-specific exported functions | 10 |
| Table 2 Tag Console Commands | 11 |
| Table 3 Exported functions of instance.c | 12 |
| Table 4: List of source files in the “dw_tag” application | 15 |
| Table 5: Field definitions within the Bink frame | 16 |
| Table 6: Table of References | 19 |
| Table 7 Document History | 20 |

2 INTRODUCTION

This document, “[DWM1004C TDoA Tag Software Guide](#)” is a guide to the application source code of the Decawave’s “dwm1004c_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 DWM1004C module, equipped by STM32L041G6U6S CORTEX-M0 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:

"dwm1004c_tdoa_tag Version 05.01.00" application version, and
"DW1000 Device Driver Version 05.01.01" driver version

The application version information can be found in source code file “[version.h](#)”.

The device driver version information can 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.

| <u>Name of file</u> | <u>Layer</u> | <u>Functional Description</u> |
|--|------------------------|--|
| 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 deca_sleep.c deca_spi.c | Target Specific Code | Code specific to reading/writing via the SPI of the target platform, I ² C, 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 STM32CubeMX and System Workbench for STM32 software (SW IDE) using the gcc compiler version 7.3.1 (7-2018-q2-update 20180622).

4.1 LOADING OF THE PROJECT TO THE IDE

Unpack the source code to the “dwm1004c_tdoa_tag” folder. In the SW IDE, choose [File->Import...](#) and import it as General/Existing project into Workspace.

4.2 CONNECTING, BUILDING AND RUNNING OF THE APPLICATION

Connect the DWM1004C-Dev 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 firmware binary into tag.

For the initial debug session you will need to create a new GDB Segger J-Link Debugging configuration in System Workbench IDE.

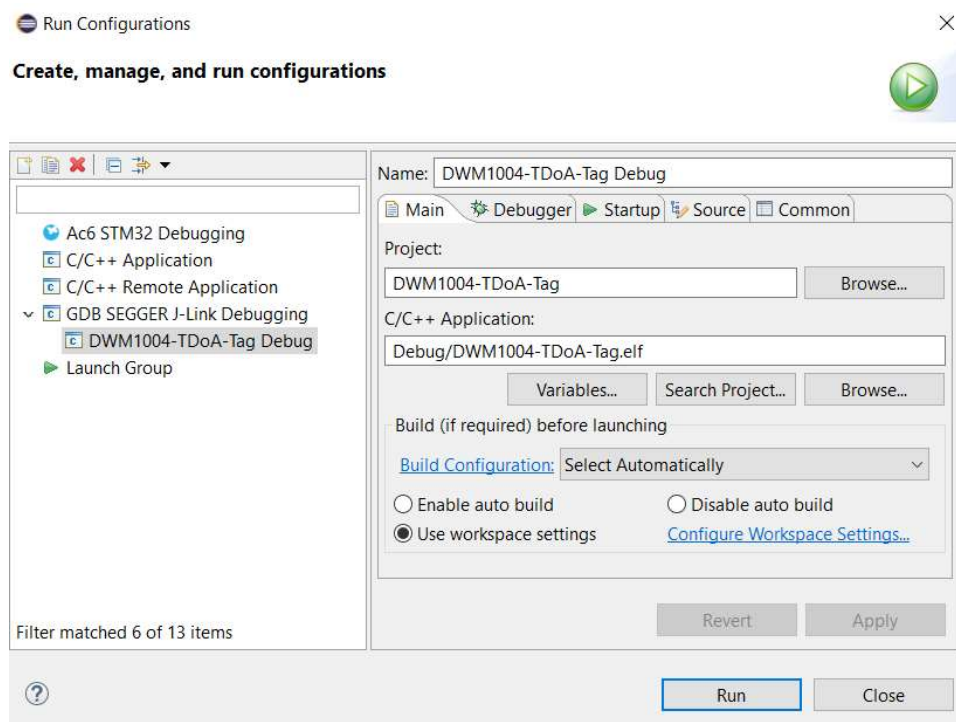


Figure 2: Debug configuration example – Main Page

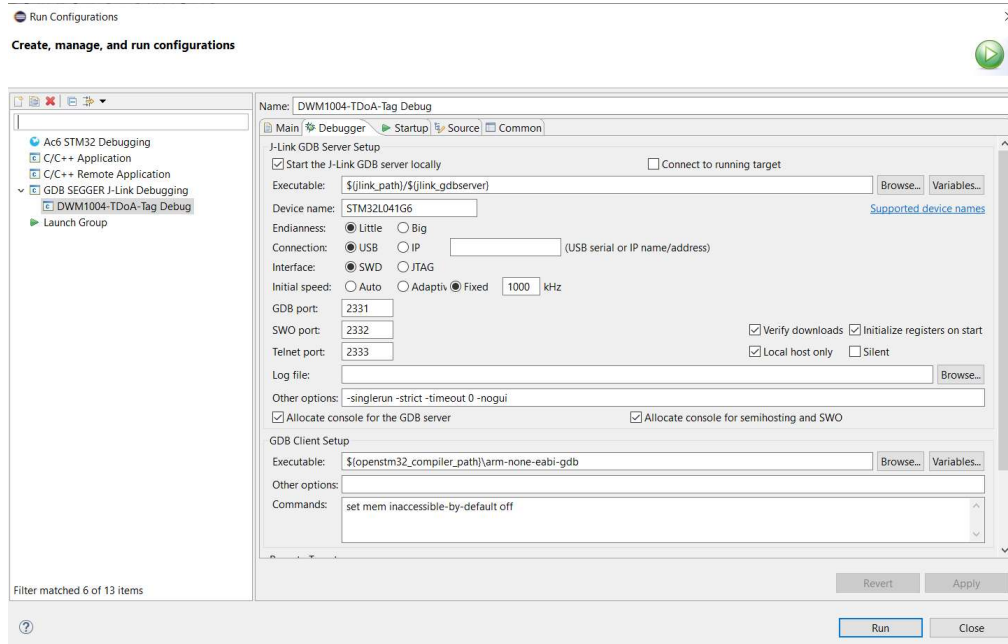


Figure 3: Debug configuration example – Debugger Page

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:

- [STMicroelectronics HAL drivers](#)
- [Decawave’s DW1000 UWB chip device driver](#)
- [Application’s target specific code](#)
- [Application itself](#)

5.1 STM DRIVERS AND LIBRARY SOURCES

The Tag hardware, DWM1004C, is based on the STMicroelectronics STM32L041G6U6S CORTEX-M0 MCU. The Hardware Abstraction Level for the TDoA Tag application is shipped within the project. See the “[\\$Project_Root/Drivers/STM32L0xx_HAL_Driver](#)”.

5.2 DECAWAVE’S DW1000 DEVICE DRIVER

Decawave’s DW1000 portable device driver can be found in the “[\\$Project_Root/Drivers/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 A – Bibliography.

5.3 Target Specific Code

The TDoA Tag application code (without HAL) 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/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”, “deca_sleep.c”, “deca_spi.c” and “deca_uart.c”

These files provide the implementation of application specific functionality to support the target hardware. This includes the Tick timer, GPIO, UART, I2C and SPI platform-dependant implementation.

Table 1 Some target-specific exported functions

| Exported function | Description |
|---|--|
| uint32 <i>portGetTickCount</i> (void) | Returns current tick count, in system clock tick time (mS) |
| void <i>port_wakeup_dw1000</i> (void) | Wakes up DW1000 chip using GPIO pins |
| void <i>port_set_dw1000_slowrate</i> (void) | Configure SPI to work on a low speed: 2 Mbit/s |
| void <i>port_set_dw1000_fastrate</i> (void) | Configure SPI to work on a fast speed: 16 Mbit/s |
| void <i>reset_DW1000</i> (void) | Toggle the reset pin of DW1000 |
| void <i>deca_uart_receive</i> (void) | Receive data from UART, stops after CR |
| void <i>port_tx_msg</i> (char * <i>buffer</i> , uint <i>len</i>) | Transmit buffer through UART |
| bool <i>deca_uart_rx_data_ready</i> () | UART command buffer is ready |
| void <i>low_power</i> (uint32 <i>ticks</i>) | Switch to stop mode, waits for ticks msec. Any configured interrupt can interrupt <i>low_power</i> () |
| int <i>readfromspi</i> (...) | DW1000 SPI read blocking function |
| int <i>writetospi</i> (...) | DW1000 SPI write blocking function |
| void <i>deca_sleep</i> (uint32 <i>ticks</i>) | DW1000 blocking wait for ticks msec |
| void <i>vTestModeMotionDetect</i> (...) | Check if need to switch between stationary and moving |
| void <i>lis3dh_configure_int</i> (...) | Setup accelerometer to generate wake-up interrupt |
| bool <i>i2c_slave_read</i> (...) | Low-level I2C read bytes |
| bool <i>i2c_slave_write</i> (...) | Low-level I2C write bytes |
| int <i>writetospi</i> (...) | DW1000 SPI write blocking function |

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 (deca_uart_rx_data_ready ())
    {
        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 38400, 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. |
| RESTORE | This command restores the NVM configuration to the factory default one. |
| RESTART | Restarts the MCU. Mandatory after any UWB parameters changed in order to apply the new settings |
| 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 <u>preamble</u> 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. |

5.5 APPLICATION: THE INSTANCE CODE

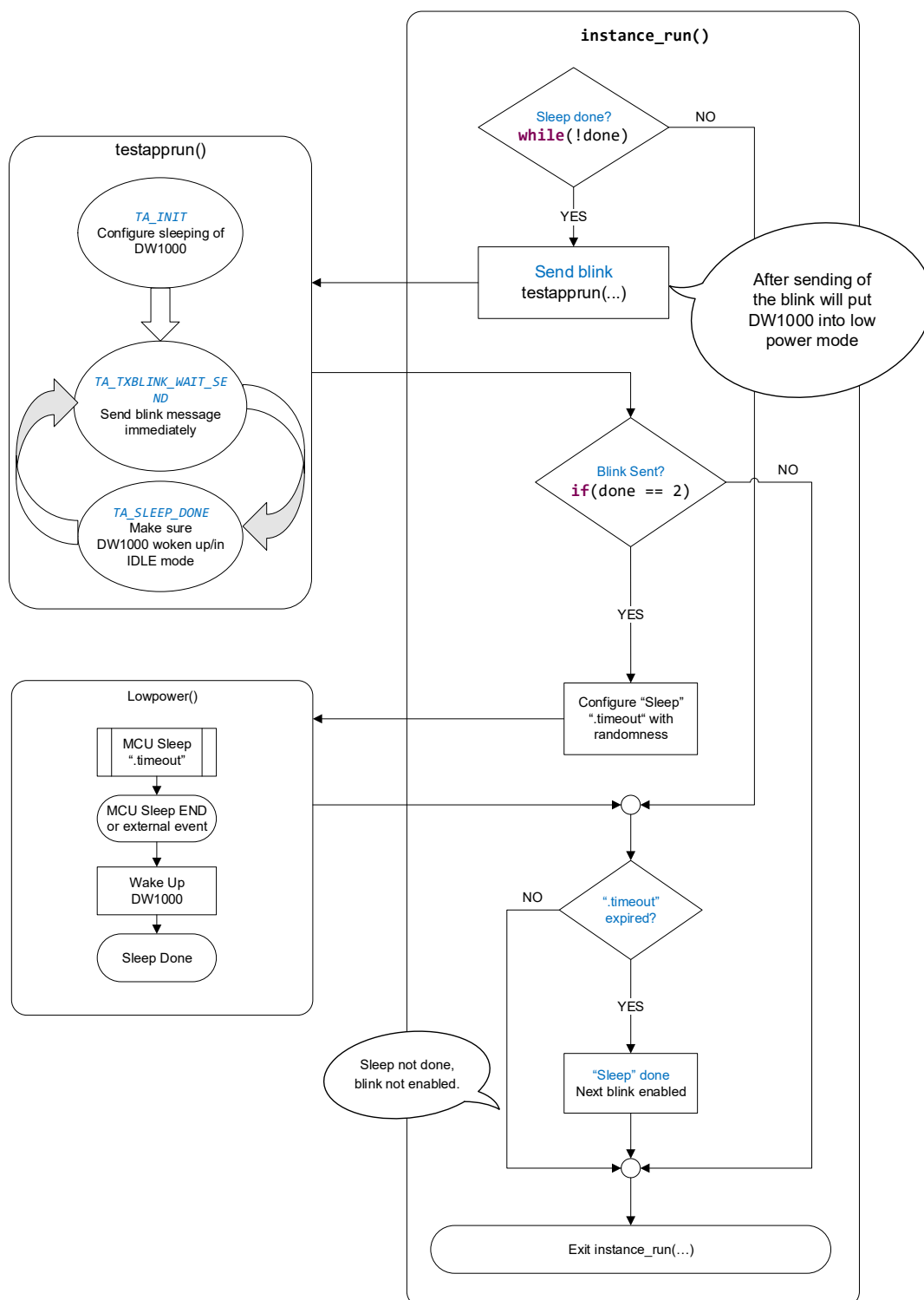
The TDoA tag application has a simple main loop organisation. The instance code, located in the [instance.c](#) provides the tag blink sending functionality, while [cmd_console.c](#) implements an interface communication to a PC terminal. The `instance_run()` has to be periodically called from a main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

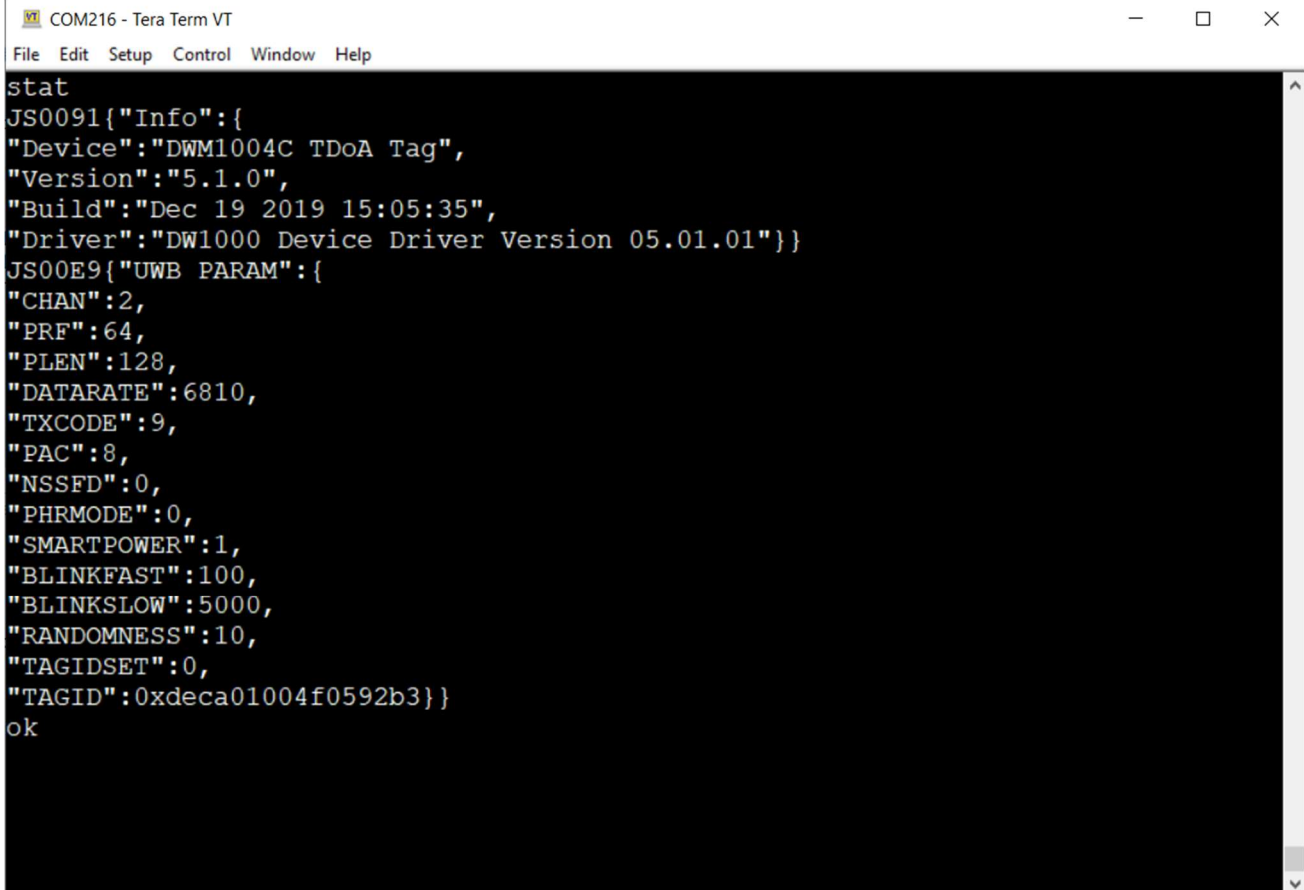
The tag application is implemented by the state machine in function [testapprun\(\)](#), called from the function `instance_run()`, which is the main entry point for running of the non-blocking process. In addition to the `instance_run()` function, the instance code provides the control functions to 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 4** below describes the operation flow of the [instance_run\(\)](#) and its interconnections with the [testapprun\(\)](#) and [lowpower\(\)](#) modules.





```
stat
JS0091{"Info":{"Device":"DWM1004C TDoA Tag",
"Version":"5.1.0",
"Build":"Dec 19 2019 15:05:35",
"Driver":"DW1000 Device Driver Version 05.01.01"}}
JS00E9{"UWB PARAM":{"CHAN":2,
"PRF":64,
"PLEN":128,
"DATARATE":6810,
"TXCODE":9,
"PAC":8,
"NSSFD":0,
"PHRMODE":0,
"SMARTPOWER":1,
"BLINKFAST":100,
"BLINKSLOW":5000,
"RANDOMNESS":10,
"TAGIDSET":0,
"TAGID":0xdeca01004f0592b3}}
ok
```

Figure 5: 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. HAL source code from STM is not included in this list but is included in the project. These files are provided by STMicroelectronics.

Table 4: List of source files in the “dw_tag” application¹

| Filename | Brief description |
|--------------------|---|
| deca_version.h | Decawave's version number for the DW1000 driver/API code |
| version.h | Application version number |
| main.c | Application – source code (main line) |
| deca_device.c | Device level Functions – source code |
| deca_device_api.h | Device level Functions – exported prototypes |
| 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 |
| cmd_uart_rx.c | Command line interface functions |
| cmd_uart_rx.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 |
| sdk_config.h | Definitions to enable peripherals used by SDK and main code |
| port_platform.c | Target-dependent functions |
| port_platform.h | Target-dependent functions prototypes |
| deca_uart.c | HW specific definitions and functions for UART Interface |
| deca_uart.h | Definitions for deca_uart.c |
| deca_spi.c | HW specific definitions and functions for SPI Interface |
| deca_spi.h | Definitions for deca_spi.c |
| translate.c | Conversions between human-readable DW1000 settings and register values (used by serial command functions) |
| translate.h | Definitions for translate.c |
| lis3dh.c | Accelerometer library |
| lis3dh.h | Exported prototypes of the accelerometer library |
| lis3dh_types.h | Constants and definitions for accelerometer library |
| lis3dh_platform.h | Platform-specific definitions |

¹ The list of files and the names of sources could be slightly different and is subject to change without notice.

5.7 TDoA TAG BLINK MESSAGE FORMAT

Only one UWB transmit packet format is employed in the Tag software, as shown in **Figure 7: Blink frame byte format**

Although these follow IEEE message conventions, this is not 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 6: 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 7: Blink frame byte format

Table 5: Field definitions within the Bink frame

| Parameter | Value | Description |
|-------------|-------|--|
| FC | 0xC5 | 1 byte frame control as per Figure 6: 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 4* 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()` state 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 7**. 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 – 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 | <p>IEEE 802.15.4-2011 or “IEEE Std 802.15.4™-2011” (Revision of IEEE Std 802.15.4-2006).</p> <p>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.</p> <p>Available from http://standards.ieee.org/</p> |
| 5 | ISO/IEC 24730-62 |

8 DOCUMENT HISTORY

Table 7 Document History

| Revision | Date | Description |
|----------|-------------|-------------------|
| 1.00 | 01-APR-2019 | First release |
| 1.01 | 23-DEC-2019 | Firmware v5.01.00 |

9 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.