



**Escuela Universitaria
Politécnica - La Almunia**
Centro adscrito
Universidad Zaragoza

**ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)**

ANEXOS

**SISTEMA DE NAVEGACIÓN PARA
VEHÍCULOS AGRÍCOLAS AUTÓNOMOS**

**NAVIGATION SYSTEM FOR
SELF-DRIVEN FARMING VEHICLES**

[424.18.58]

Autor: Joaquín Pérez Sancho

Director: Jesús Ponce de León Vázquez

Fecha: 28/11/2018



INDICE DE CONTENIDO

ANEXO 1. PROGRAMAS	2
1.1. PROGRAMA OFICIAL DEL SISTEMA	2
1.2. PROGRAMA DE GUARDADO DE LA TRAZADA	7
1.3. PROGRAMA DE LECTURA Y NAVEGACIÓN	10
ANEXO 2. MÓDULO DWM1001-DEV	16

ANEXO 1. PROGRAMAS

1.1. PROGRAMA OFICIAL DEL SISTEMA

```
#include "dwm_api.h"

#include "hal.h"

#include "stdio.h"

#include "string.h"

#include <math.h>

void position(float*,float*);

void setup(){

    dwm_cfg_tag_t cfg_tag;

    dwm_cfg_t cfg_node;

    HAL_Print("dwm_init(): dev%d\n", HAL_DevNum());

    dwm_init();

    HAL_Print("Setting to tag: dev%d.\n", HAL_DevNum());

    cfg_tag.low_power_en = 0;

    cfg_tag.meas_mode = DWM_MEAS_MODE_TWR;

    cfg_tag.loc_engine_en = 1;

    cfg_tag.common.led_en = 1;

    cfg_tag.common.ble_en = 1;

    cfg_tag.common.uwb_mode = DWM_UWB_MODE_ACTIVE;

    cfg_tag.common.fw_update_en = 0;

    HAL_Print("dwm_cfg_tag_set(&cfg_tag): dev%d.\n", HAL_DevNum());

    dwm_cfg_tag_set(&cfg_tag);
```



```
HAL_Print("Wait 2s for node to reset.\n");

HAL_Delay(2000);

dwm_cfg_get(&cfg_node);

HAL_Print("Comparing set vs. get: dev%d.\n", HAL_DevNum());

if((cfg_tag.low_power_en      != cfg_node.low_power_en)
|| (cfg_tag.meas_mode        != cfg_node.meas_mode)
|| (cfg_tag.loc_engine_en    != cfg_node.loc_engine_en)
|| (cfg_tag.common.led_en    != cfg_node.common.led_en)
|| (cfg_tag.common.ble_en    != cfg_node.common.ble_en)
|| (cfg_tag.common.uwb_mode  != cfg_node.common.uwb_mode)
|| (cfg_tag.common.fw_update_en != cfg_node.common.fw_update_en))
{
    HAL_Print("low_power_en          cfg_tag=%d : cfg_node=%d\n",
cfg_tag.low_power_en,  cfg_node.low_power_en);

    HAL_Print("meas_mode            cfg_tag=%d : cfg_node=%d\n",
cfg_tag.meas_mode,    cfg_node.meas_mode);

    HAL_Print("loc_engine_en        cfg_tag=%d : cfg_node=%d\n",
cfg_tag.loc_engine_en,  cfg_node.loc_engine_en);

    HAL_Print("common.led_en         cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.led_en,  cfg_node.common.led_en);

    HAL_Print("common.ble_en           cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.ble_en,  cfg_node.common.ble_en);

    HAL_Print("common.uwb_mode        cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.uwb_mode,  cfg_node.common.uwb_mode);

    HAL_Print("common.fw_update_en     cfg_tag=%d :  cfg_node=%d\n",
cfg_tag.common.fw_update_en,  cfg_node.common.fw_update_en);

    HAL_Print("\nConfiguration failed.\n\n");
}

else
```

programas

```
{
    HAL_Print("\nConfiguration succeeded.\n\n");
}

}

FILE*f;

int main(void)
{
    setup();

    int k=1, mode=0;
    char pos[20];
    char j[100];
    char d[100];
    float x,y,xt,yt,xv,yv,mod=2,alpha;
    const char s[2] = ",";
    char *token;
    char d1,d2;

    printf("Escoja el tipo de modo a utilizar:\n");
    printf("MODO LECTURA = 1\nMODO ESCRITURA = 2\n");
    scanf("%d",&mode);
    if (mode == 1)
    {
        f=fopen("track.txt","r");
        while(fgets(pos,20,f)!=NULL)
        {
            printf("La posicion de la trazada es: %s",pos);
            token =strtok(pos,s);
```

```
xt= atof(token);

token = strtok(NULL,s);

yt=atof(token);

while(mod>1.00) // Offset para indicar a partir de cuando estamos dentro de
la posicion
{
    position(&x,&y);

    printf("La posición del vehículo es: %f, %f\n",x,y);

xv = xt-x;
yv = yt-y;

printf("El vector a seguir es: %f,%f \n",xv,yv);

mod =sqrt(pow(xv,2)+pow(yv,2));

printf("La distancia a recorrer: %f \n",mod);

alpha = atan(yv/xv)*(180/3.141592);//lo pasamos a grados

if(yv>0){
d1 = 'N';
} if(yv<0){
d1='S';
}if(yv==0.0){
d2 = ' ';
}

if(xv>0){
d2='E';
} if(xv<0){
d2='O';
}

if(xv==0.0){
```

programas

```
d2 = ' ';
}

printf("La orientacion es de: %f grados en direccion: %c%c\n\n",alpha,d1,d2);

}

}

}

if (mode == 2)
{
    f=fopen("track.txt","wt");
    while(k<10)
    {
        k=k+1;

        //scanf("%d",&k);// Aqui debemos de poner el iniciador del hardware de

        HAL_Delay(1000);          // cuando empezar a grabar y cuando no.

        position(&x,&y);

        printf("La posicion del vehiculo es: %f, %f\n",x,y);

        sprintf(j,"%f",x);

        fputs(j,f);

        sprintf(d,"%f\n",y);

        fputs(d,f);

    }
}

fclose(f);

return 0;

}
```



```
void position(float* x, float* y){
    dwm_loc_data_t loc;

    dwm_pos_t pos;

    loc.p_pos = &pos;

    if(dwm_loc_get(&loc) == RV_OK)
    {
        *x=loc.p_pos->x;
        *y=loc.p_pos->y;
    }
}
```

1.2. PROGRAMA DE GUARDADO DE LA TRAZADA

```
#include "dwm_api.h" //Libreria para poder usar el modulo DWM1001-DEV
#include "hal.h"
#include "stdio.h"
#include "stdlib.h"

FILE*f; //Puntero utilizado para el uso de las funciones de fichero

int main(void){

    int i, k=1;

    float x,y;

    char s[100], d[100];

    int wait_period = 1000;

    dwm_cfg_tag_t cfg_tag;

    dwm_cfg_t cfg_node;

    f = fopen("track.txt","wt"); //Se abre el fichero en modo escribir

    HAL_Print("dwm_init(): dev%d\n", HAL_DevNum());

    dwm_init();
```

programas

```
    HAL_Print("Setting to tag: dev%d.\n", HAL_DevNum());

    cfg_tag.low_power_en = 0;

    cfg_tag.meas_mode = DWM_MEAS_MODE_TWR;

    cfg_tag.loc_engine_en = 1;

    cfg_tag.common.led_en = 1;

    cfg_tag.common.ble_en = 1;

    cfg_tag.common.uwb_mode = DWM_UWB_MODE_ACTIVE;

    cfg_tag.common.fw_update_en = 0;

    HAL_Print("dwm_cfg_tag_set(&cfg_tag):dev%d.\n",HAL_DevNum());

    dwm_cfg_tag_set(&cfg_tag);

    HAL_Print("Wait 2s for node to reset.\n");

    HAL_Delay(2000);

    dwm_cfg_get(&cfg_node);

    HAL_Print("Comparing set vs. get: dev%d.\n", HAL_DevNum());

    if((cfg_tag.low_power_en      != cfg_node.low_power_en)

    || (cfg_tag.meas_mode        != cfg_node.meas_mode)

    || (cfg_tag.loc_engine_en    != cfg_node.loc_engine_en)

    || (cfg_tag.common.led_en    != cfg_node.common.led_en)

    || (cfg_tag.common.ble_en    != cfg_node.common.ble_en)

    || (cfg_tag.common.uwb_mode  != cfg_node.common.uwb_mode)

    || (cfg_tag.common.fw_update_en      !=

    cfg_node.common.fw_update_en))

    {

        HAL_Print("low_power_en      cfg_tag=%d      :      cfg_node=%d\n",

        cfg_tag.low_power_en,      cfg_node.low_power_en);

        HAL_Print("meas_mode        cfg_tag=%d        :      cfg_node=%d\n",

        cfg_tag.meas_mode,        cfg_node.meas_mode);
```

```
    HAL_Print("loc_engine_en      cfg_tag=%d      :      cfg_node=%d\n",
cfg_tag.loc_engine_en,          cfg_node.loc_engine_en);
HAL_Print("common.led_en      cfg_tag=%d      :      cfg_node=%d\n",
cfg_tag.common.led_en,        cfg_node.common.led_en);
HAL_Print("common.ble_en      cfg_tag=%d      :      cfg_node=%d\n",
cfg_tag.common.ble_en,        cfg_node.common.ble_en);
HAL_Print("common.uwb_mode    cfg_tag=%d      :      cfg_node=%d\n",
cfg_tag.common.uwb_mode,      cfg_node.common.uwb_mode);
HAL_Print("common.fw_update_en  cfg_tag=%d      :      cfg_node=%d\n",
cfg_tag.common.fw_update_en,  cfg_node.common.fw_update_en);

    HAL_Print("\nConfiguration failed.\n\n");
}
else
{
    HAL_Print("\nConfiguration succeeded.\n\n");
}
dwm_loc_data_t loc;

dwm_pos_t pos;

loc.p_pos = &pos;

while(k==1)
{
    scanf("%d",&k);// Aqui debemos de poner el iniciador del hardware de
                    // cuando grabar y cuando parar de hacerlo.

    HAL_Print("Wait %d ms...\n", wait_period);

    HAL_Delay(wait_period);

    HAL_Print("dwm_loc_get(&loc):\n");

    if(dwm_loc_get(&loc) == RV_OK)
    {
```

programas

```
    HAL_Print("\t[%d,%d,%u]\n", loc.p_pos->x, loc.p_pos->y,  
             loc.p_pos->qf);  
  
    x=loc.p_pos->x;  
  
    sprintf(s,"%f",x); //Se pasa el valor de la coordenada "x" de tipo float a  
//tipo char para guardarlo en el fichero. Junto con una coma para crear un token.  
  
    fputs(s,f); //Se guarda en el fichero  
  
    y=loc.p_pos->y;  
  
    sprintf(d,"%f\n",y); // Se pasa el valor de la coordenada "y" de tipo float a  
//tipo char para guardarlo en el fichero. Junto con un final de linea o "intro"  
//para que la siguiente coordenada la escriba bajo.  
  
    fputs(d,f); //Se guarda en el fichero  
  
    }  
  
    }  
  
    fclose(f); //Se cierra correctamente el fichero para que guarde los datos.  
  
    return 0;  
  
}
```

1.3. PROGRAMA DE LECTURA Y NAVEGACIÓN

```
#include "dwm_api.h"  
  
#include "hal.h"  
  
#include "stdio.h"  
  
#include "string.h"  
  
#include "math.h"  
  
  
void position(float*,float*); // Se declara una función y de qué tipo es  
  
void setup(){ // Configuración del modulo UWB  
  
    dwm_cfg_tag_t cfg_tag;  
  
    dwm_cfg_t cfg_node;  
  
    HAL_Print("dwm_init(): dev%d\n", HAL_DevNum());
```

```
dwm_init();

HAL_Print("Setting to tag: dev%d.\n", HAL_DevNum());

cfg_tag.low_power_en = 0;

cfg_tag.meas_mode = DWM_MEAS_MODE_TWR;

cfg_tag.loc_engine_en = 1;

cfg_tag.common.led_en = 1;

cfg_tag.common.ble_en = 1;

cfg_tag.common.uwb_mode = DWM_UWB_MODE_ACTIVE;

cfg_tag.common.fw_update_en = 0;

HAL_Print("dwm_cfg_tag_set(&cfg_tag):dev%d.\n",HAL_DevNum());

dwm_cfg_tag_set(&cfg_tag);

HAL_Print("Wait 2s for node to reset.\n");

HAL_Delay(2000);

dwm_cfg_get(&cfg_node);

HAL_Print("Comparing set vs. get: dev%d.\n", HAL_DevNum());

if((cfg_tag.low_power_en    != cfg_node.low_power_en)
|| (cfg_tag.meas_mode      != cfg_node.meas_mode)
|| (cfg_tag.loc_engine_en  != cfg_node.loc_engine_en)
|| (cfg_tag.common.led_en  != cfg_node.common.led_en)
|| (cfg_tag.common.ble_en  != cfg_node.common.ble_en)
|| (cfg_tag.common.uwb_mode != cfg_node.common.uwb_mode)
|| (cfg_tag.common.fw_update_en !=
cfg_node.common.fw_update_en))
{
    HAL_Print("low_power_en    cfg_tag=%d : cfg_node=%d\n",
cfg_tag.low_power_en,    cfg_node.low_power_en);

    HAL_Print("meas_mode      cfg_tag=%d : cfg_node=%d\n",
cfg_tag.meas_mode,      cfg_node.meas_mode);
```

programas

```
        HAL_Print("loc_engine_en      cfg_tag=%d : cfg_node=%d\n",
cfg_tag.loc_engine_en,  cfg_node.loc_engine_en);

        HAL_Print("common.led_en      cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.led_en,  cfg_node.common.led_en);

        HAL_Print("common.ble_en      cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.ble_en,  cfg_node.common.ble_en);

        HAL_Print("common.uwb_mode    cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.uwb_mode, cfg_node.common.uwb_mode);

        HAL_Print("common.fw_update_en cfg_tag=%d : cfg_node=%d\n",
cfg_tag.common.fw_update_en, cfg_node.common.fw_update_en);

        HAL_Print("\nConfiguration failed.\n\n");
    }
    else
    {
        HAL_Print("\nConfiguration succeeded.\n\n");
    }
}

FILE*f; // Puntero utilizado para el uso del fichero

int main(void)
{
    setup(); //Se llama a la función Setup para ejecutar la configuración del
modulo

    char pos[20];

    float x,y,xt,yt,xv,yv,mod=2,alpha;

    const char s[2] = ","; // Constante coma que separa las dos coordenadas.

    char *token; // Puntero tipo char para usar en la funcion token

    char d1,d2;

    f=fopen("track.txt","r"); // Se abre el fichero en modo lectura

    while(fgets(pos,20,f)!=NULL) // Bucle en el que se lee las posiciones
hasta //que no quede ninguna en el fichero
```

```

{
    printf("La posicion de la trazada es: %s",pos);

    token =strtok(pos,s); // Guardamos en la variable token el valor de la
primera //coordenada "x"

    xt= atof(token); // Guarda el valor de la primera coordenada en tipo float

    token = strtok(NULL,s); // Pasamos ahora el valor de la coordenada "y" a
la //variable token

    yt=atof(token); //Guarda el valor de la coordenada en tipo float.

    while(mod>1.00) //Bucle. Offset para indicar a partir de cuando estamos
//dentro de la posición
    {
        position(&x,&y); //Llamada a la función que calcula la posición instantánea
del //vehiculo

        printf("La posicion del vehiculo es: %f, %f\n",x,y);

        xv = xt-x; // Cálculo de la coordenada "x" del vector mediante resta
        yv = yt-y; // Cálculo de la coordenada "y" del vector mediante resta
        printf("El vector a seguir es: %f,%f \n",xv,yv);

        mod =sqrt(pow(xv,2)+pow(yv,2)); // cálculo del modulo del vector a
seguir

        printf("La distancia a recorrer: %f \n",mod);

        alpha = atan(yv/xv)*(180/3.141592);//Cálculo del ángulo de giro en
grados

        // #Calculo de que dirección tomar (N-norte, S- sur, E-este, O-oeste) # //
        if(yv>0){
            d1 = 'N';
        } if(yv<0){
            d1='S';
        }if(yv==0.0){

```

programas

```
    d2 = ' ';
}
if(xv>0){
    d2='E';
} if(xv<0){
    d2='O';
}
if(xv==0.0){
    d2 = ' ';
}
printf("La orientación es de: %f grados en direccion:
%c%c\n\n",alpha,d1,d2);

}
mod=2;
}
fclose(f); // Se cierra correctamente el fichero para evitar errores
return 0;
}

void position(float* x, float* y){ // Función que calcula la posición
instantánea //del vehículo.
    int i;
    int wait_period = 1000;
    dwm_loc_data_t loc;
    dwm_pos_t pos;
    loc.p_pos = &pos;
    HAL_Delay(wait_period);
```



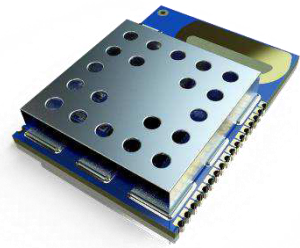

```
if(dwm_loc_get(&loc) == RV_OK)
{
    *x=loc.p_pos->x; // Pasamos el valor de la coordenada "x" mediante
función.
    *y=loc.p_pos->y; // Pasamos el valor de la coordenada "y" mediante
función.
}
}
```

ANEXO 2. MÓDULO DWM1001-DEV



Overview

The DWM1001 module is based on Decawave's DW1000 Ultra Wideband (UWB) transceiver IC, which is an IEEE 802.15.4-2011 UWB implementation. It integrates UWB and Bluetooth® antenna, all RF circuitry, Nordic Semiconductor nRF52832 and a motion sensor.

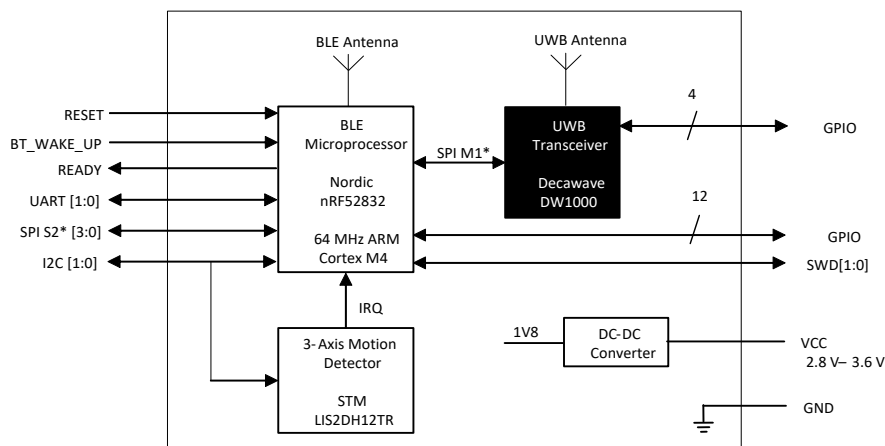


Key Features

- Ranging accuracy to within 10cm.
- UWB Channel 5 printed PCB antenna (6.5 GHz)
- 6.8 Mbps data rate
- 60 m line-of-sight range typical
- IEEE 802.15.4-2011 UWB compliant
- Nordic Semiconductor nRF52832
- Bluetooth® connectivity
- Bluetooth® chip antenna
- Motion sensor: 3-axis accelerometer
- Current consumption optimised for low power sleep mode: <math><15\mu\text{A}</math>
- Supply voltage: 2.8 V to 3.6 V
- Size: 19.1 mm x 26.2 mm x 2.6 mm

Key Benefits

- Enables anchors, tags & gateways to quickly get an entire RTLS system up-and-running
- Accelerates product designs for faster Time-to-Market & reduced development costs
- Ready-to-go embedded firmware to minimise software development
- Over-the-air updates
- User API to DWM1001 firmware (available as a library) for user code customisation
- On-board *Bluetooth*® SMART for connectivity to phones/tablets/PCs
- SPI, UART and *Bluetooth*® APIs to access DWM1001 firmware from an external device
- Low-power hardware design and software architecture for longer battery life



*SPI M1 is nRF52 SPI master 1, SPI S2 is SPI slave 2

High Level Block Diagram

Table of Contents

1 OVERVIEW	5	5 TRANSMIT AND RECEIVE POWER CONSUMPTION	13
1.1 UWB TRANSCEIVER DW1000	5	6 ANTENNA PERFORMANCE	14
1.2 BLUETOOTH® MICROPROCESSOR NORDIC NRF52832	5	7 APPLICATION INFORMATION.....	16
1.3 POWER SUPPLY AND POWER MANAGEMENT.....	5	7.1 APPLICATION BOARD LAYOUT GUIDELINES.....	16
1.4 THREE AXIS MOTION DETECTOR STMICROELECTRONICS LIS2DH12TR	5	8 PACKAGE INFORMATION	17
1.5 SOFTWARE ON BOARD	5	8.1 MODULE DRAWINGS.....	17
2 DWM1001 CALIBRATION	6	8.2 MODULE LAND PATTERN	18
2.1.1 <i>Crystal Oscillator Trim</i>	6	8.3 MODULE MARKING INFORMATION	18
2.1.2 <i>Transmitter Calibration</i>	6	8.4 MODULE SOLDER PROFILE.....	19
2.1.3 <i>Antenna Delay Calibration</i>	6	9 ORDERING INFORMATION	20
3 DWM1001 PIN CONNECTIONS	7	9.1 TAPE AND REEL INFORMATION	20
3.1 PIN NUMBERING	7	10 GLOSSARY	21
3.2 PIN DESCRIPTIONS.....	7	11 REFERENCES	22
4 ELECTRICAL SPECIFICATIONS	10	12 DOCUMENT HISTORY	22
4.1 NOMINAL OPERATING CONDITIONS	10	13 MAJOR CHANGES	22
4.2 DC CHARACTERISTICS.....	10	14 ABOUT DECAWAVE	23
4.3 RECEIVER AC CHARACTERISTICS	10		
4.4 RECEIVER SENSITIVITY CHARACTERISTICS	11		
4.5 TRANSMITTER AC CHARACTERISTICS	11		
4.5.1 <i>Absolute Maximum Ratings</i>	12		

List of Figures

FIGURE 1: DWM1001 PIN DIAGRAM	7	FIGURE 5: MODULE PACKAGE SIZE (UNITS: MM).....	17
FIGURE 2: POWER CONSUMPTION DURING TWO WAY RANGING.....	13	FIGURE 6: DWM1001 MODULE LAND PATTERN (UNITS: MM).....	18
FIGURE 3. ANTENNA RADIATION PATTERN PLANES	14	FIGURE 7: DWM1001 MODULE SOLDER PROFILE.....	19
FIGURE 4: DWM1001 APPLICATION BOARD KEEP-OUT AREAS.....	16	FIGURE 8: DWM1001 TAPE AND REEL DIMENSIONS	20

List of Tables

TABLE 1: DWM1001 PIN FUNCTIONS	7
TABLE 2: EXPLANATION OF ABBREVIATIONS.....	9
TABLE 3: INTERNAL NRF52832 PINS USED AND THEIR FUNCTION	9
TABLE 4: I2C SLAVE DEVICES ADDRESSI2C.....	9
TABLE 5: DWM1001 OPERATING CONDITIONS	10
TABLE 6: DWM1001 DC CHARACTERISTICS	10
TABLE 7: DWM1001 RECEIVER AC CHARACTERISTICS ...	10
TABLE 8: DWM1001 TYPICAL RECEIVER SENSITIVITY CHARACTERISTICS	11
TABLE 9: DWM1001 TRANSMITTER AC CHARACTERISTICS	11
TABLE 10: DWM1001 ABSOLUTE MAXIMUM RATINGS.	12
TABLE 11. ANTENNA RADIATION PATTERNS	15
TABLE 12: GLOSSARY OF TERMS.....	21
TABLE 13: DOCUMENT HISTORY.....	22

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 with Decawave for the most recent updates on this product.

The DWM1001 is pre-loaded with firmware, please refer to the "DWM1001 Firmware User Guide" for disclaimer and license terms.

Copyright © 2017 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.

REGULATORY APPROVALS

The DWM1001, as supplied from Decawave, has not been certified for use in any particular geographic region by the appropriate regulatory body governing radio emissions in that region although it is capable of such certification depending on the region and the manner in which it is used.

All products developed by the user incorporating the DWM1001 must be approved by the relevant authority governing radio emissions in any given jurisdiction prior to the marketing or sale of such products in that jurisdiction and user bears all responsibility for obtaining such approval as needed from the appropriate authorities.

1 OVERVIEW

The block diagram on page 1 of this data sheet shows the major sections of the DWM1001. An overview of these blocks is given below.

1.1 UWB Transceiver DW1000

The module has a DW1000 UWB transceiver mounted on the PCB. The DW1000 uses a 38.4 MHz reference crystal. The crystal has been trimmed in production to reduce the initial frequency error to approximately 3 ppm, using the DW1000 IC's internal on-chip crystal trimming circuit.

Always-On (AON) memory can be used to retain DW1000 configuration data during the lowest power operational states when the on-chip voltage regulators are disabled. This data is uploaded and downloaded automatically. Use of DW1000 AON memory is configurable.

The on-chip voltage and temperature monitors allow the host to read the voltage on the VDDAON pin and the internal die temperature information from the DW1000.

See the DW1000 Datasheet [2] for more detailed information on device functionality, electrical specifications and typical performance.

1.2 Bluetooth® Microprocessor Nordic nRF52832

The nRF52832 is an ultra-low power 2.4 GHz wireless system on chip (SoC) integrating the nRF52 Series 2.4 GHz transceiver and an ARM Cortex-M4 CPU with 512kB flash memory and 64kB RAM.

See the nRF52832 Datasheet[1] for more detailed information on device functionality, electrical specifications and typical performance.

1.3 Power Supply and Power management

The power management circuit consists of a switch mode regulator. It is a buck convertor or step down convertor. The input voltage to the DWM1001 can be in the range 2.8V to 3.6V. Outputs from the convertor provides 1.8V which is required by the DW1000[2]

1.4 Three Axis Motion Detector STMicroelectronics LIS2DH12TR

The LIS2DH12 is an ultra-low-power high performance three-axis linear accelerometer with digital I2C/SPI serial interface standard output. The LIS2DH12 has user-selectable full scales of $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ and is capable of measuring accelerations with output data rates from 1 Hz to 5.3 kHz. The self-test capability allows the user to check the functionality of the sensor in the final application. The device may be configured to generate interrupt signals by detecting two independent inertial wake-up/free-fall events as well as by the position of the device itself. The LIS2DH12 is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

See the LIS2DH12TR Datasheet[4] for more detailed information on device functionality, electrical specifications and typical performance.

1.5 Software on board

The DWM1001 module comes pre-loaded with embedded firmware which provides two-way ranging (TWR) and real time location system (RTLS) functionality. See the details in the DWM1001 Firmware User Guide [6]. The module can be configured and controlled via its API, which can be accessed through a number of different interfaces, allowing flexibility to the product designer. The details of the API are described in the DWM1001 Firmware API Guide [5]. Decawave also provides the module firmware in the form of binary libraries and some source code. A build environment is provided, so that the user can customise the operation and if required add their own functions[6].

2 DWM1001 CALIBRATION

Depending on the end-use applications and the system design, DWM1001 settings may need to be tuned. To help with this tuning a number of built in functions such as continuous wave TX and continuous packet transmission can be enabled. See the DW1000 User Manual [3] for further details.

2.1.1 Crystal Oscillator Trim

DWM1001 modules are calibrated at production to minimise initial frequency error to reduce carrier frequency offset between modules and thus improve receiver sensitivity. The calibration carried out at module production will trim the initial frequency offset to less than 3 ppm, typically.

2.1.2 Transmitter Calibration

The DWM1001 is calibrated in module production for the on board firmware application. This is calibrated to meet the power spectral density requirement of less than -41.3 dBm/MHz.

2.1.3 Antenna Delay Calibration

In order to measure range accurately, precise calculation of timestamps is required. To do this the antenna delay must be known. The DWM1001 allows this delay to be calibrated and provides the facility to compensate for delays introduced by PCB, external components, antenna and internal DWM1001 delays.

If using the pre-loaded embedded firmware of the DWM1001 module, the Antenna Delay has been pre calibrated for this configuration. The antenna delay is stored in OTP memory.

If you are creating your own embedded firmware, with a different configuration for the DW1000, then you will have to calibrate antenna delay. To calibrate the antenna delay, range is measured at a known distance using two DWM1001 systems. Antenna delay is adjusted until the known distance and reported range agree.

Antenna delay calibration must be carried out as a once off measurement for each DWM1001 design implementation. If required, for greater accuracy, antenna delay calibration should be carried out on a per DWM1001 module basis, see DW1000 User Manual [3] for full details. Further details can be found in the Antenna Delay Application Note [8].

3 DWM1001 PIN CONNECTIONS

3.1 Pin Numbering

DWM1001 module pin assignments are as follows (viewed from top): -

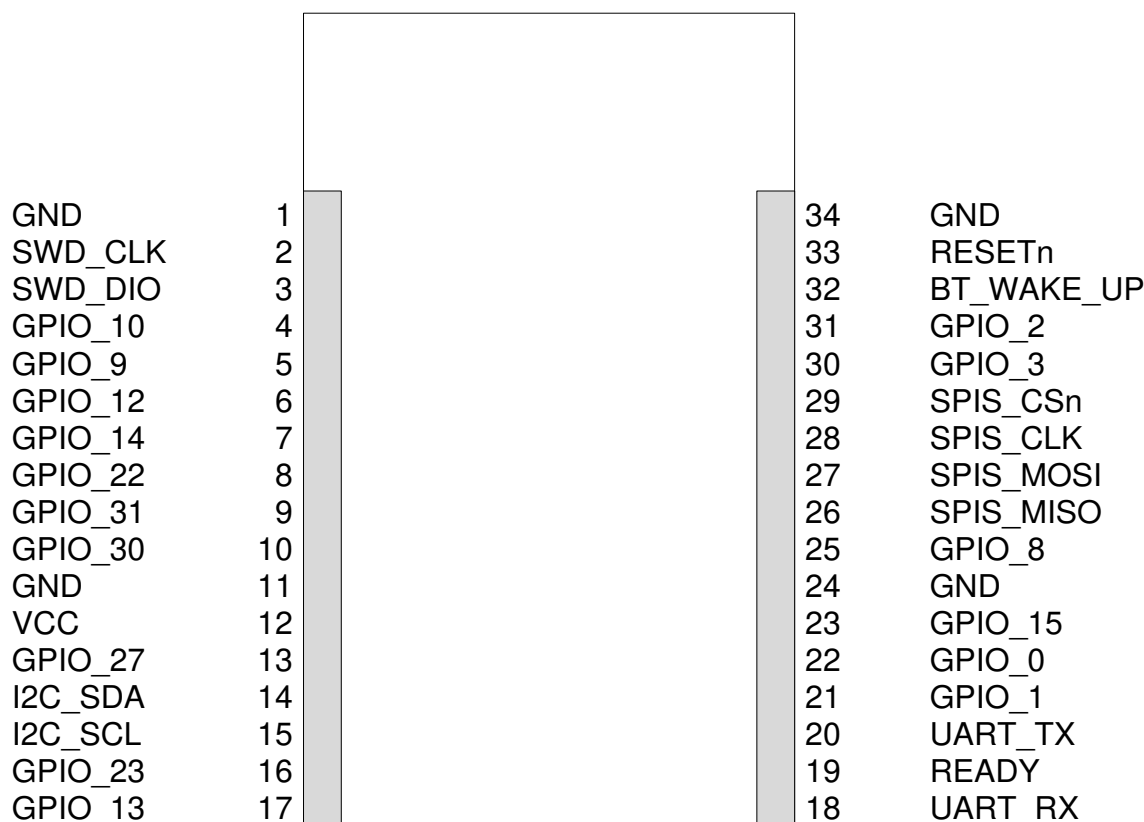


Figure 1: DWM1001 Pin Diagram

3.2 Pin Descriptions

Pin details are given in

Table 1: DWM1001 Pin functions

SIGNAL NAME	PI N	I/O (Default)	DESCRIPTION	REFERENCE (Pin designation)
Digital Interface				
SWD_CLK	2	DI	Serial wire debug clock input for debug and programming of Nordic Processor	[N] SWDCLK
SWD_DIO	3	DIO	Serial wire debug I/O for debug and programming of Nordic Processor	[N] SWDIO
GPIO_10	4	DIO	General purpose I/O pin.	[N] P0.10
GPIO_9	5	DIO	General purpose I/O pin.	[N] P0.9
GPIO_12	6	DIO	General purpose I/O pin.	[N] P0.12
GPIO_14	7	DIO	General purpose I/O pin.	[N] P0.14
GPIO_22	8	DIO	General purpose I/O pin.	[N] P0.22
GPIO_31	9	DIO	General purpose I/O pin. ADC function of nRF52	[N] P0.31
GPIO_30	10	DIO	General purpose I/O pin. ADC function of nRF52	[N] P0.30
GPIO_27	13	DIO	General purpose I/O pin.	[N] P0.27
I2C_SDA (Master)	14	DIO	Master I2C Data Line.	[N] P0.29

SIGNAL NAME	PI N	I/O (Default)	DESCRIPTION	REFERENCE (Pin designation)
I2C_SCL (Master)	15	DO	Master I2C Clock Line	[N] P0.28
GPIO_23	16	DIO	General purpose I/O pin.	[N] P0.23
GPIO_13	17	DIO	General purpose I/O pin.	[N] P0.13
UART_RX	18	DI	UART_RX	[N] P0.11
READY	19	DO	Generated interrupt from the device. Indicates events such as SPI data ready, or location data ready. See the function <code>dwm_int_cfg()</code> in the DWM1001 Firmware API Guide for details[5].	[N] P0.26
UART_TX	20	DO	UART_TX, This is also the ADC function of the nRF52	[N] P0.05
GPIO_1	21	DIO	General purpose I/O pin of the DW1000. It may be configured for use as a SFDLED driving pin that can be used to light a LED when SFD (Start Frame Delimiter) is found by the receiver. Refer to the DW1000 User Manual [1] for details of LED use.	[DW] GPIO1
GPIO_0	22	DIO	General purpose I/O pin of the DW1000. It may be configured for use as a RXOKLED driving pin that can be used to light a LED on reception of a good frame. Refer to the DW1000 User Manual [1] for details of LED use.	[DW] GPIO0
GPIO_15	23	DIO	General purpose I/O pin.	[N] P0.15
GPIO_8	25	DIO	General purpose I/O pin.	[N] P0.08
SPIS_MISO	26	DI	Configured as a SPI slave this pin is the SPI data output. Refer to Datasheet for more details [1].	[N] P0.07
SPIS_MOSI	27	DO	Configured as a SPI slave this pin is the SPI data input. Refer to Datasheet for more details [1].	[N] P0.06
SPIS_CLK	28	DI	Configured as a SPI slave this pin is the SPI clock. This is also the ADC function of the nRF52	[N] P0.04
SPIS_CSn	29	DI	Configured as a SPI slave this pin is the SPI chip select. This is an active low enable input. The high-to-low transition on SPICSn signals the start of a new SPI transaction. This is also the ADC function of the nRF52	[N] P0.03
GPIO_3	30	DO	This pin is configured for use as a TXLED driving pin that can be used to light a LED during transmit mode. Refer to the DW1000 User Manual [2] for details of LED use.	[DW] GPIO3
GPIO_2	31	DO	This pin is configured for use as a RXLED driving pin that can be used to light a LED during receive mode. Refer to the DW1000 User Manual [2] for details of LED use.	[DW] GPIO2
BT_WAKE_UP	32	DI	When this pin is asserted to its active low state the Bluetooth device will advertise its availability for 20 seconds by broadcasting advertising packets. This is also the ADC function of the nRF52.	[N] P0.02
RESETn	33	DI	Reset pin. Active Low Input.	[N] P0.21
Power Supplies				
VCC	12	P	External supply for the module. 2.8V - 3.6V	

SIGNAL NAME	PIN	I/O (Default)	DESCRIPTION	REFERENCE (Pin designation)
Ground				
GND	1, 11, 24, 34	G	Common ground.	

Table 2: Explanation of Abbreviations

ABBREVIATION	EXPLANATION
DI	Digital Input
DIO	Digital Input / Output
DO	Digital Output
G	Ground
P	Power Supply
N	nRF52832
DW	DW1000

Note: Any signal with the suffix 'n' indicates an active low signal.

Table 3: Internal nRF52832 pins used and their function

nRF52832 Pin	Function
PO.19	DW_IRQ
PO.16	DW_SCK
PO.20	DW_MOSI
PO.18	DW_MISO
PO.17	DW_SPI_CS
PO.24	DW_RST
PO.25	ACC_IRQ
PO.29	I2C_SDA
PO.28	I2C_SCL

DW1000's GPIOs 5,6 control the DW1000 SPI mode configuration. Within the DWM1001 module, those GPIOs are unconnected and will be internally pulled down. Consequently, SPI will be set to mode 0. For more details, please refer to DW1000 data sheet [2].

Table 4: I2C slave devices address I2C

I2C slave device	Address
LIS2DH12	0X19

4 ELECTRICAL SPECIFICATIONS

The following tables give detailed specifications for the DWM1001 module. $T_{amb} = 25\text{ }^{\circ}\text{C}$ for all specifications given.

4.1 Nominal Operating Conditions

Table 5: DWM1001 Operating Conditions

Parameter	Min.	Typ.	Max.	Units	Condition/Note
Operating temperature	-40		+85	$^{\circ}\text{C}$	
Supply voltage VCC	2.8	3.3	3.6	V	Normal operation
Voltage on VDDIO for programming OTP	3.7	3.8	3.9	V	Note that for programming the OTP in the DWM1001 this supply is connected to the VDDIO test point which is underneath the PCB. (See Figure 6)

4.2 DC Characteristics

Table 6: DWM1001 DC Characteristics

Parameter	Min.	Typ.	Max.	Units	Condition/Note
Supply current in DEEP SLEEP mode		4		μA	All peripherals in lowest power consumption mode Achievable where RTC and accelerometer are disabled with custom firmware.
Supply current in DEEP SLEEP mode		12		μA	RTC and accelerometer operational, all other peripherals in lowest power consumption mode*
Supply current in IDLE mode		13		mA	MCU and DW1000 awake
TX peak current		111		mA	
TX mean current		82		mA	
RX peak current		154		mA	
RX mean current		134		mA	
Current in Bluetooth® discovery mode		6		mA	
Digital input voltage high	$0.7 \times \text{VCC}$		VCC	V	
Digital input voltage low	GND		$0.3 \times \text{VCC}$	V	
Digital output voltage high	$0.7 \times \text{VCC}$		VCC	V	
Digital output voltage low	GND		$0.3 \times \text{VCC}$	V	

* Using a ranging update rate of 1 Hz

4.3 Receiver AC Characteristics

Table 7: DWM1001 Receiver AC Characteristics

Parameter	Min.	Typ.	Max.	Units	Condition/Note
Frequency range	6240		6739.2	MHz	Centre Frequency 6489.6 MHz

4.4 Receiver Sensitivity Characteristics

$T_{amb} = 25\text{ }^{\circ}\text{C}$, 20 byte payload. These sensitivity figures assume an antenna gain of 0 dBi and should be modified by the antenna characteristics, depending on the orientation of the DWM1001.

Table 8: DWM1001 Typical Receiver Sensitivity Characteristics

Packet Error Rate	Data Rate	Receiver Sensitivity	Units	Condition/Note		
1%	6.8 Mbps	-98*(-92)	dBm/500 MHz	Preamble 128	Carrier frequency offset ± 10 ppm	All measurements performed on Channel 5, PRF 64 MHz
10%	6.8 Mbps	-99*(-93)	dBm/500 MHz	Preamble 128		

*equivalent sensitivity with Smart TX Power enabled. This is enabled in the onboard firmware.

4.5 Transmitter AC Characteristics

Table 9: DWM1001 Transmitter AC Characteristics

Parameter	Min.	Typ.	Max.	Units	Condition/Note
Frequency range	6240		6739.2	MHz	
Output power spectral density			-41.3*	dBm/MHz	See DW1000 Datasheet [1]
Output Channel Power		-17		dBm/500MHz	
Output power variation with temperature*	-1		+1	dB	Using on board compensation.

* If using the pre-loaded embedded firmware of the DWM1001 module, otherwise see the DW1000 datasheet

4.5.1 Absolute Maximum Ratings

Table 10: DWM1001 Absolute Maximum Ratings

Parameter	Min.	Max.	Units
Supply voltage	2.8	3.9	V
Receiver power		0	dBm
Temperature - Storage temperature	-40	+85	°C
Temperature – Operating temperature	-40	+85	°C
ESD (Human Body Model)		2000	V
DWM1001 pins other than VCC, VDDIO and GND		3.6	Note that 3.6 V is the max voltage that may be applied to these pins

Stresses beyond those listed in this table may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions beyond those indicated in the operating conditions of the specification is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

5 TRANSMIT AND RECEIVE POWER CONSUMPTION

The following Figures give power profiles for the DWM1001 on a DWM1001-DEV PCB when used for Two Way Ranging, see Figure 2. Peak values are given.

Figure 2 shows an example of the power consumption of a DWM1001 tag running the factory loaded firmware.

The tag is in low-power mode, and two-way ranging with 3 anchors. The deep-sleep current occurs while the tag is sleeping with only the RTC and accelerometer active.

Once awake, the tag transmits at its allocated time in the TDMA-slotting, and awaits the anchors responses. This can be observed as 1 transmission followed by 3 receives, repeated once. After this is completed, the tag spends some time computing its location, before returning to sleep. The total time awake is dependent on the number of anchors within range of the tag. For more details on the system operation, see the DWM1001 System Overview document[9].

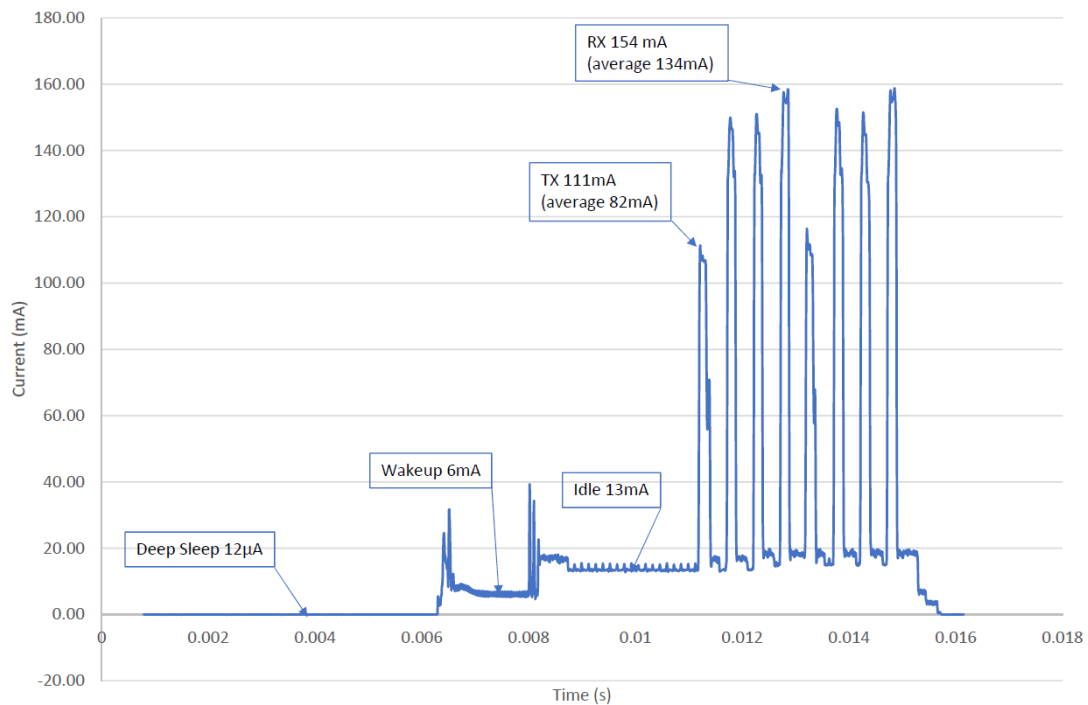


Figure 2: power consumption during Two Way Ranging

6 ANTENNA PERFORMANCE

This section details antenna radiation patterns for the DWM1001-Dev board. Figure 3 presents a view of the measurement planes considered in this document.

Table 11 shows antenna radiation patterns for the DWM1001 module mounted on the DWM1001-Dev board. Three planes in the spherical space about the centre of the board are measured, with theta and phi plots representing perpendicular polarisations.

The DWM1001 antenna is vertically polarised, meaning that the module is intended to be positioned vertically upright when used in an RTLS system. An omnidirectional radiation pattern is seen in the XZ plane when observed by another antenna which is also vertically polarised. This is shown in the XZ plane antenna patterns, where the vertically polarised plot, phi, has a circular, or omnidirectional shape.

If the antennas are oriented perpendicular relative to each other, then the polarisation changes. In this case, the horizontally polarised pattern, theta, applies and there are nulls at certain angles which can limit range and introduce location inaccuracy.

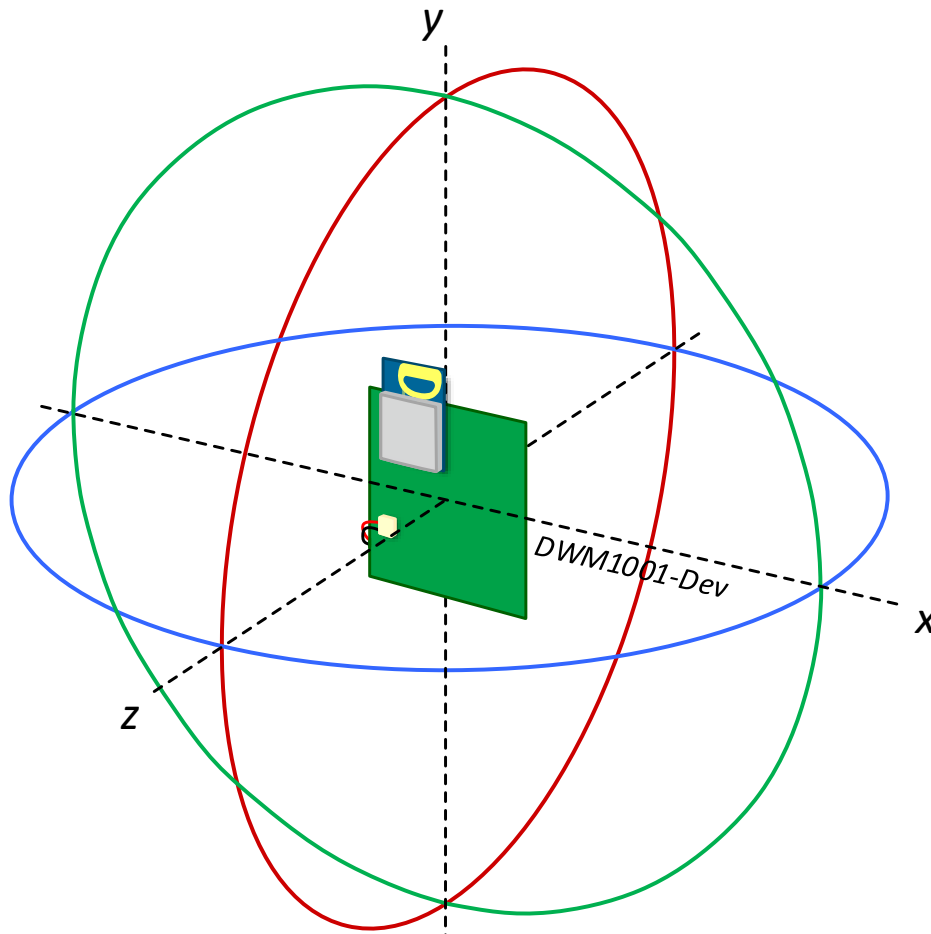
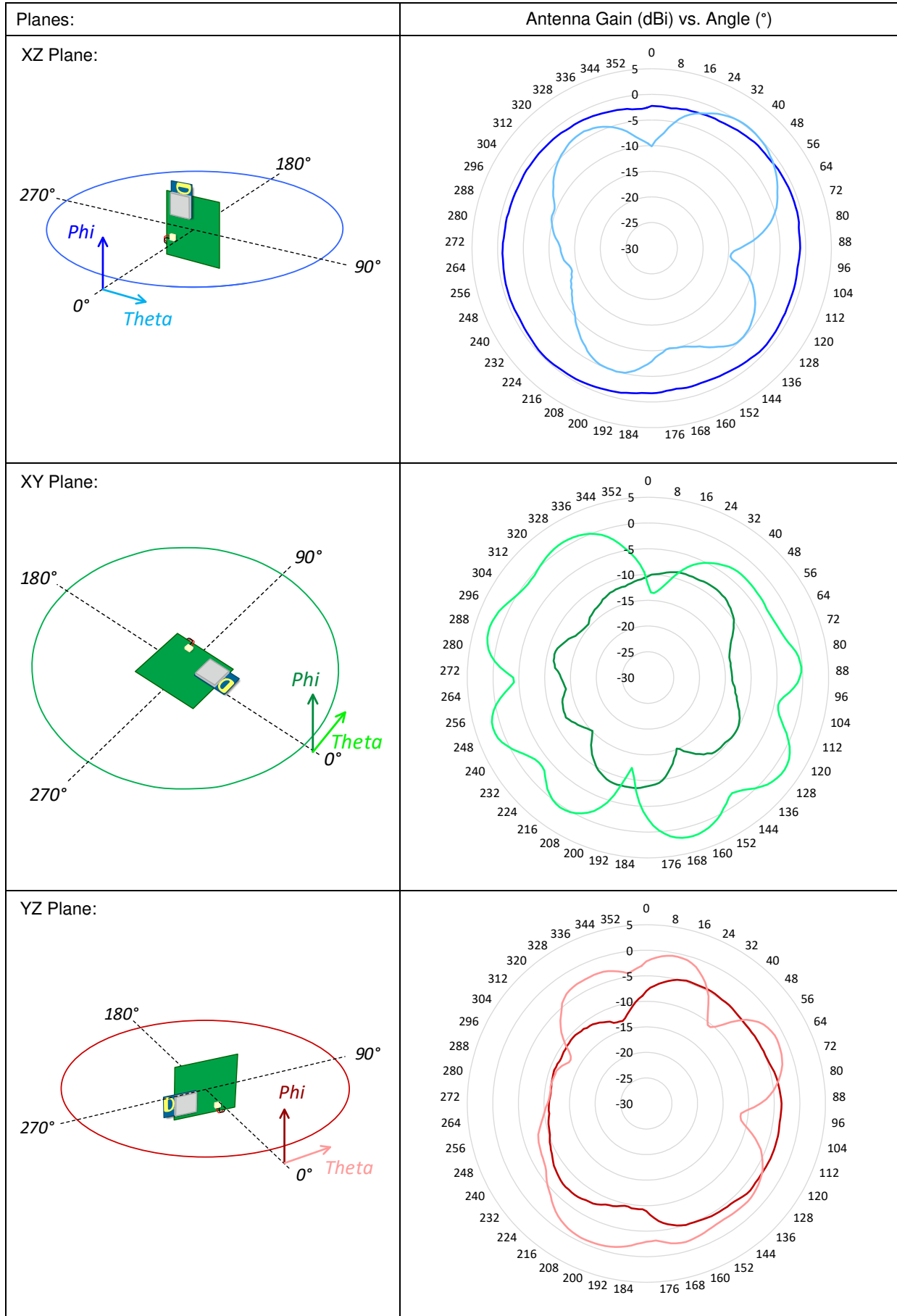


Figure 3. Antenna Radiation Pattern Planes

Table 11. Antenna Radiation Patterns



7 APPLICATION INFORMATION

7.1 Application Board Layout Guidelines

When designing the PCB onto which the DWM1001 will be soldered, the proximity of the DWM1001 on-board antenna to metal and other non-RF transparent materials needs to be considered carefully. Two suggested placement schemes are shown below.

For best RF performance, ground copper should be flooded in all areas of the application board, except in the areas marked “Keep-Out Area”, where there should be no metal either side, above or below (e.g. do not place battery under antenna).

The two placement schemes in **Figure 4** show an application board with no metallic material in the keep-out area. The diagram on the right is an application board with the antenna projecting off of the board so that the keep out area is in free-space. The diagram on the left shows an application board which does not have the module in free space but has the pcb copper removed on either side (and behind) the module antenna.

(Note: the rectangular area above the shield on the module is the antenna area)

It is also important to note that the ground plane on the application board affects the DWM1001 antenna radiation pattern. There must be a minimum spacing of 10 mm (d) without metal either side of the module antenna.

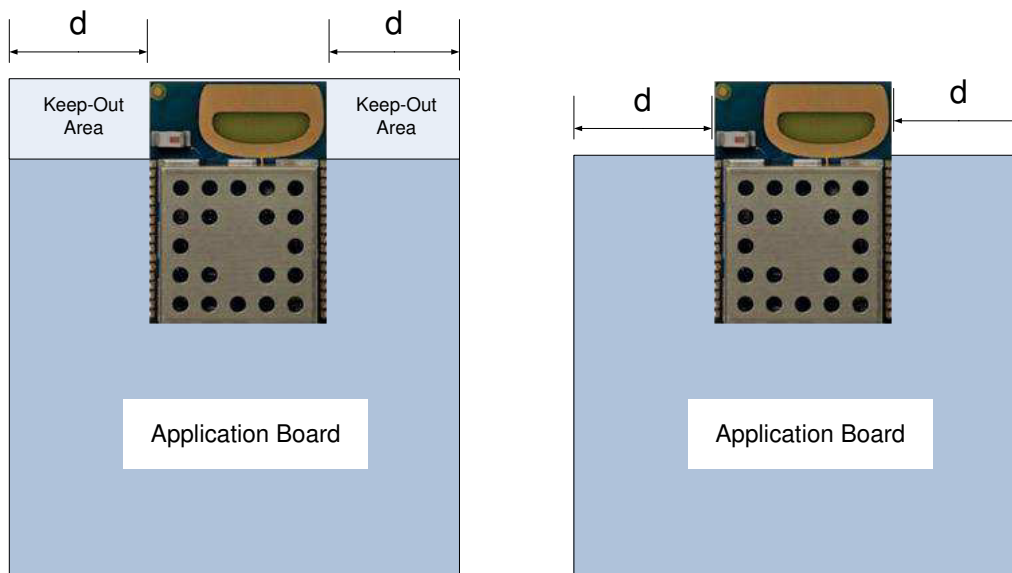


Figure 4: DWM1001 Application Board Keep-Out Areas

8 PACKAGE INFORMATION

8.1 Module Drawings

All measurements are given in millimetres.

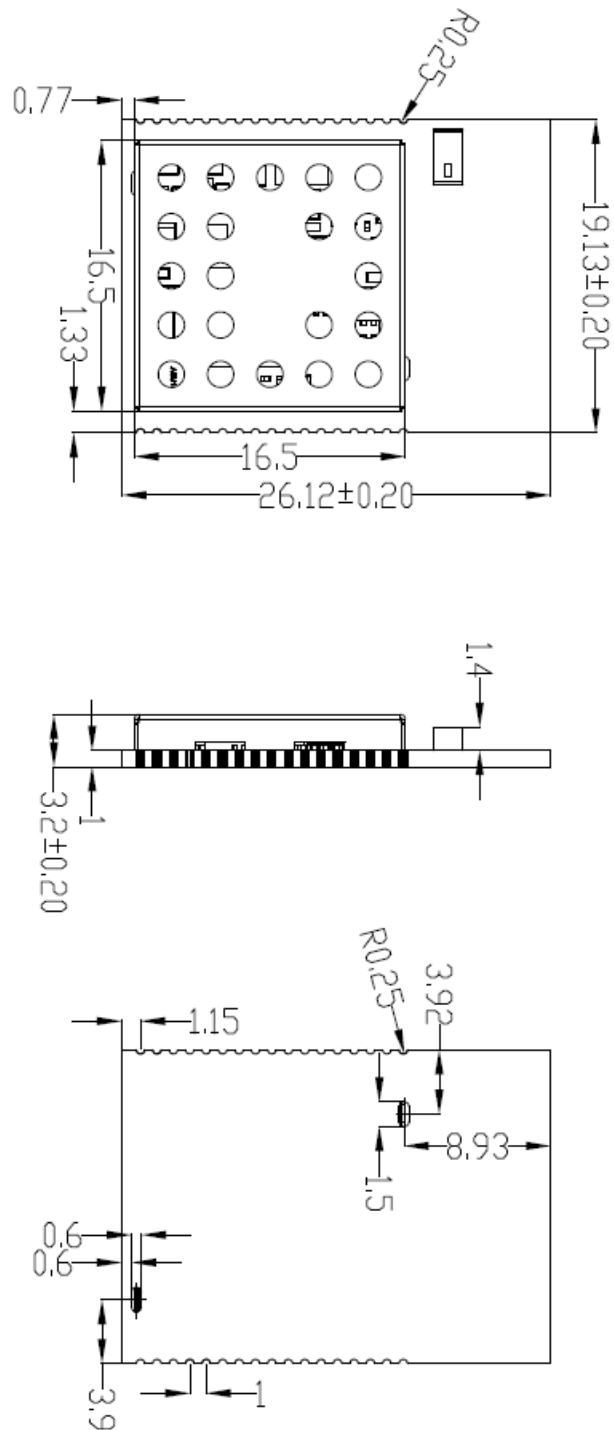


Figure 5: Module Package Size (units: mm)

8.2 Module Land Pattern

The diagram below shows the DWM1001 module land pattern.

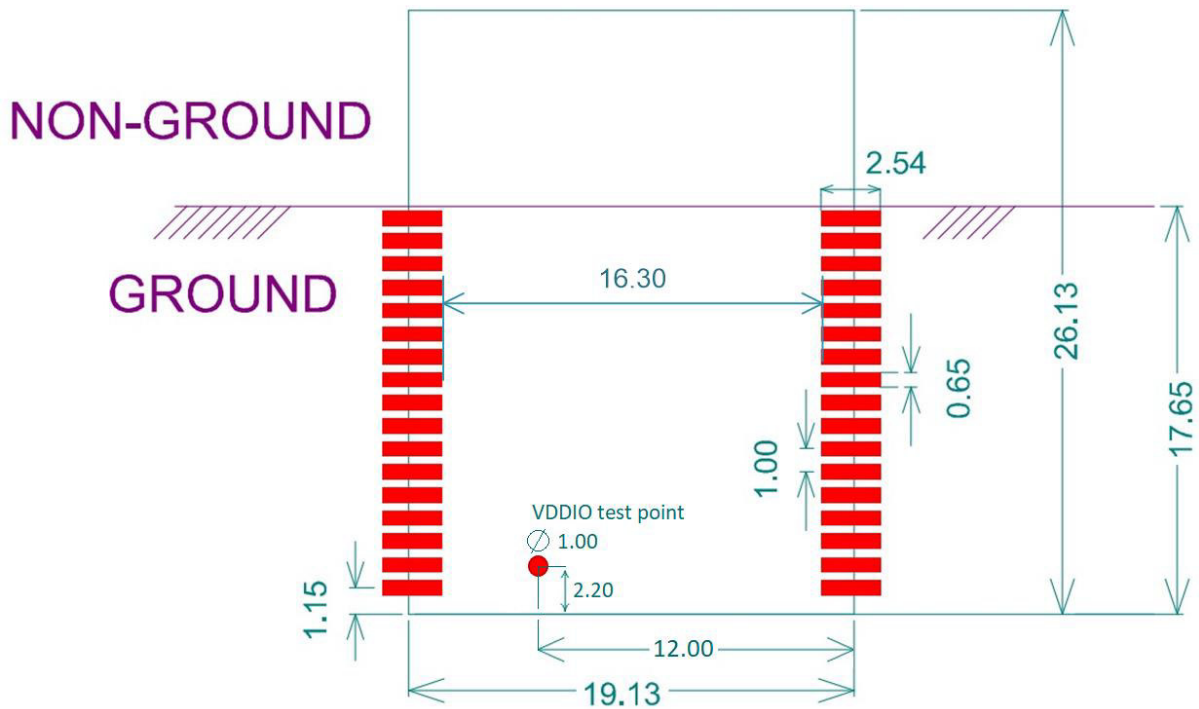


Figure 6: DWM1001 Module Land Pattern (units: mm)

8.3 Module Marking Information

Each module has a label on the shield with a serial number in the following format:

YY WW 0 SSSSS

Where:

YY	indicates the year
WW	indicates the week of the year
0	indicates the DWM1001 module
SSSSS	indicates the module manufacturing number

8.4 Module Solder Profile

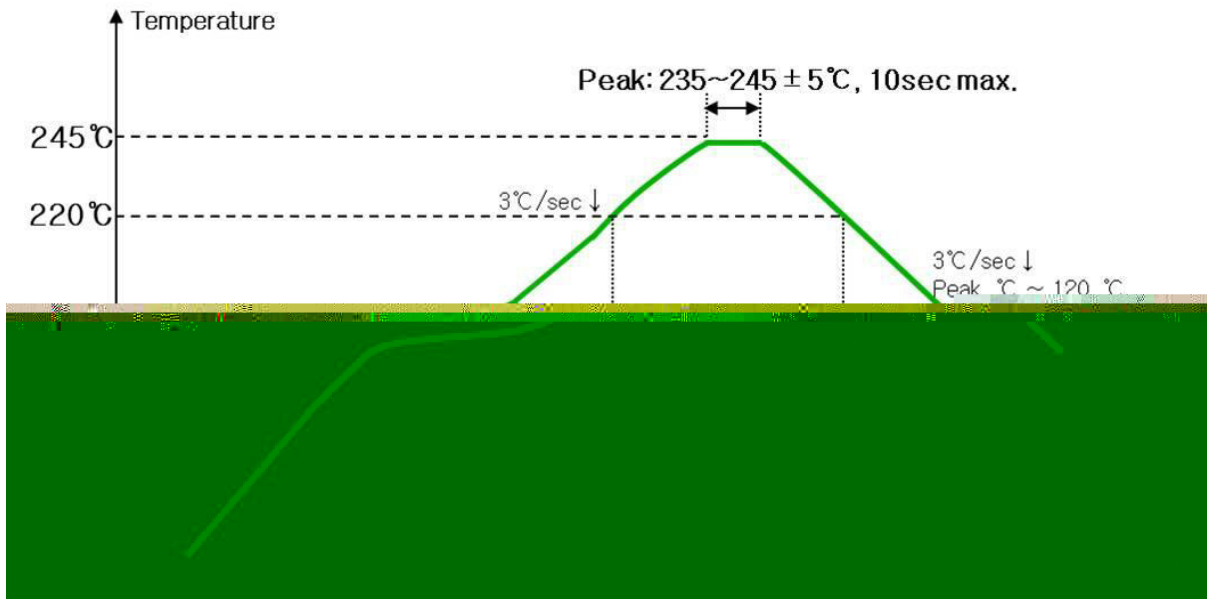


Figure 7: DWM1001 Module Solder Profile

9 ORDERING INFORMATION

9.1 Tape and Reel Information

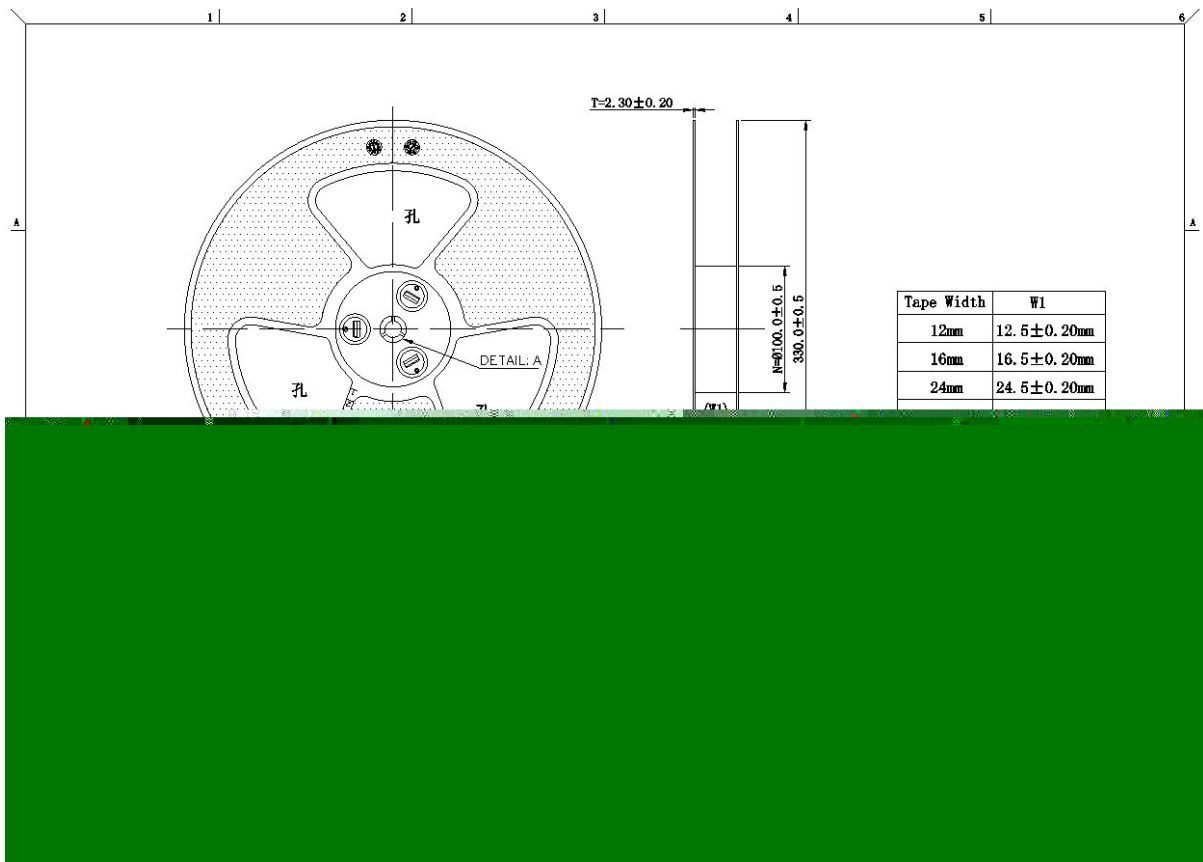


Figure 8: DWM1001 Tape and Reel Dimensions

10 GLOSSARY

Table 12: Glossary of Terms

Abbreviation	Full Title	Explanation
EIRP	Equivalent Isotropically Radiated Power	The amount of power that a theoretical isotropic antenna (which evenly distributes power in all directions) would emit to produce the peak power density observed in the direction of maximum gain of the antenna being used
ETSI	European Telecommunication Standards Institute	Regulatory body in the EU charged with the management of the radio spectrum and the setting of regulations for devices that use it
FCC	Federal Communications Commission	Regulatory body in the USA charged with the management of the radio spectrum and the setting of regulations for devices that use it
GPIO	General Purpose Input / Output	Pin of an IC that can be configured as an input or output under software control and has no specifically identified function
IEEE	Institute of Electrical and Electronic Engineers	Is the world's largest technical professional society. It is designed to serve professionals involved in all aspects of the electrical, electronic and computing fields and related areas of science and technology
LIFS	Long Inter-Frame Spacing	Defined in the context of the IEEE 802.15.4-2011 [7] standard
LNA	Low Noise Amplifier	Circuit normally found at the front-end of a radio receiver designed to amplify very low level signals while keeping any added noise to as low a level as possible
LOS	Line of Sight	Physical radio channel configuration in which there is a direct line of sight between the transmitter and the receiver
NLOS	Non Line of Sight	Physical radio channel configuration in which there is no direct line of sight between the transmitter and the receiver
PGA	Programmable Gain Amplifier	Amplifier whose gain can be set / changed via a control mechanism usually by changing register values
PLL	Phase Locked Loop	Circuit designed to generate a signal at a particular frequency whose phase is related to an incoming "reference" signal.
PPM	Parts Per Million	Used to quantify very small relative proportions. Just as 1% is one out of a hundred, 1 ppm is one part in a million
RF	Radio Frequency	Generally used to refer to signals in the range of 3 kHz to 300 GHz. In the context of a radio receiver, the term is generally used to refer to circuits in a receiver before down-conversion takes place and in a transmitter after up-conversion takes place
RTLS	Real Time Location System	System intended to provide information on the location of various items in real-time.
SFD	Start of Frame Delimiter	Defined in the context of the IEEE 802.15.4-2011 [7] standard.
SPI	Serial Peripheral Interface	An industry standard method for interfacing between IC's using a synchronous serial scheme first introduced by Motorola
TCXO	Temperature Controlled Crystal Oscillator	A crystal oscillator whose output frequency is very accurately maintained at its specified value over its specified temperature range of operation.
TWR	Two Way Ranging	Method of measuring the physical distance between two radio units by exchanging messages between the units and noting the times of transmission and reception. Refer to Decawave's website for further information
TDOA	Time Difference of Arrival	Method of deriving information on the location of a transmitter. The time of arrival of a transmission at two physically different locations whose clocks are synchronized is noted and the difference in the arrival times provides information on the location of the transmitter. A number of such TDOA measurements at different locations can be used to uniquely determine the position of the transmitter. Refer to Decawave's website for further information.
UWB	Ultra Wideband	A radio scheme employing channel bandwidths of, or in excess of, 500MHz
WSN	Wireless Sensor Network	A network of wireless nodes intended to enable the monitoring and control of the physical environment
BLE	Bluetooth Low Energy.	A low power means of data communication.

11 REFERENCES

- [1] nRF52832 Product Specification v1.3 www.nordicsemi.com
- [2] Decawave DW1000 Datasheet www.decawave.com
- [3] Decawave DW1000 User Manual www.decawave.com
- [4] STMicroelectronics LIS2DH12TR www.st.com
- [5] DWM1001 Firmware API Guide
- [6] DWM1001 Firmware User Guide
- [7] IEEE802.15.4-2011 or "IEEE Std 802.15.4™-2011" (Revision of IEEE Std 802.15.4-2006). IEEE Standard for Local and metropolitan area networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Computer Society Sponsored by the LAN/MAN Standards Committee. Available from <http://standards.ieee.org/>
- [8] APS014 Antenna Delay Calibration of DW1000-based products and systems
- [9] DWM1001 System Overview

12 DOCUMENT HISTORY

Table 13: Document History

Revision	Date	Description
1.0	21/12/17	First release
1.10	27/02/18	Update

13 MAJOR CHANGES

Revision 1.10

Page	Change Description
All	Update of version number to 1.10
9	New table detailing internal connections between nRF52 and DW1000
9	Adding I2C slave devices address
9	Specifying that nRF52 to DW1000 SPI interface mode is 0
14,15	New details on Antenna Radiation pattern.
18	Adding accurate position of VDDIO test point on figure 6

14 ABOUT DECAWAVE

Decawave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4-2011 [7] UWB standard. This device is the first in a family of parts that will operate at data rates of 110 kbps, 850 kbps, 6.8 Mbps.

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

Further Information

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

Decawave Ltd
Adelaide Chambers
Peter Street
Dublin
D08 T6YA
Ireland
+353 1 6975030
e: sales@decawave.com
w: www.decawave.com



**DWM1001 Firmware User
Guide**

Based on DWM1001-DEV board

Version 1.1

**This document is subject to change without
notice**

TABLE OF CONTENTS

DISCLAIMER	5
1 INTRODUCTION	8
1.1 OVERVIEW	8
1.2 WHAT'S INCLUDED IN THE FIRMWARE RELEASE.....	8
2 FIRMWARE OVERVIEW	9
2.1 HIGH-LEVEL ARCHITECTURE	9
2.2 OVERVIEW OF THE PANS LIBRARY	10
2.3 COMPONENTS AND OPERATIONS OF THE PANS LIBRARY.....	10
2.3.1 <i>SoftDevice and BLE driver</i>	11
2.3.2 <i>eCos RTOS and BSP</i>	11
2.3.3 <i>DW1000 driver</i>	11
2.3.4 <i>Stationary indication</i>	11
2.3.5 <i>DRTLS Network operation</i>	11
2.3.6 <i>Commissioning</i>	11
2.3.7 <i>RTLS Management</i>	11
2.3.8 <i>TWR solver / Location Engine</i>	12
3 FIRMWARE TOOL CHAIN	13
3.1 TOOL CHAIN OVERVIEW	13
3.2 CONTENT IN THE TOOL CHAIN.....	13
3.2.1 <i>Hardware part of toolchain</i>	14
3.2.2 <i>Software part of toolchain</i>	14
3.2.3 <i>Example application package for DWM1001</i>	14
3.3 GUIDES TO FLASH THE DWM1001 WITH FACTORY IMAGE	14
3.3.1 <i>Using J-Flash Light</i>	14
3.3.2 <i>Using the provided VM tools</i>	15
4 USER APPLICATION EXAMPLES	18
4.1 OVERVIEW	18
4.2 "C CODE API" APPLICATION EXAMPLE	18
4.2.1 <i>Firmware image partitioning</i>	18
4.2.2 <i>Compiling/debugging user application in the firmware</i>	19
4.3 UART APPLICATIONS EXAMPLE.....	21
4.3.1 <i>UART connection</i>	21
4.3.2 <i>UART examples</i>	23
4.4 SPI APPLICATIONS EXAMPLE.....	25
4.4.1 <i>SPI connection</i>	25
4.4.2 <i>SPI example</i>	25
5 REFERENCES	27
6 DOCUMENT HISTORY	28
6.1 REVISION HISTORY	28
7 CHANGE LOG	29
8 ABOUT DECAWAVE	30

LIST OF TABLES

TABLE 1 UART PIN CONNECTIONS.....	22
TABLE 2 SPI PIN CONNECTIONS.....	25
TABLE 3: DOCUMENT HISTORY	28

LIST OF FIGURES

FIGURE 1 HIGH-LEVEL ARCHITECTURE OF DWM1001 FIRMWARE VS. USER SOFTWARE.....	9
FIGURE 2 DWM1001 FIRMWARE LIBRARYCOMPONENTS	10
FIGURE 3 TOOL CHAIN AND SOURCE COMPONENTS IN DWM1001 FIRMWARE DEVELOPMENT.....	13
FIGURE 4 DWM1001 DEV BOARD – MICRO USB CONNECTION.....	15
FIGURE 5 TRANSFER DWM1001 FROM WINDOWS TO VIRTUALBOX.....	17
FIGURE 6 DWM1001 FLASH ADDRESS MAP.....	18
FIGURE 7 J-LINK DEVICE IN WINDOWS.....	22
FIGURE 8 DWM1001 DEVICE COM PORT OVER J-LINK	22
FIGURE 9 CONNECTING DWM1001 TO RASPBERRY PI 3 OVER HEADER PINS	23
FIGURE 10 CONNECT TO DWM1001 DEVICE THROUGH UART SHELL	24
FIGURE 11 RUN SIMPLE UART EXAMPLE ON RASPBERRY PI 3	25
FIGURE 12 RUN SIMPLE SPI EXAMPLE ON RASPBERRY PI 3	26

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 © 2017 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

- (1) This Disclaimer applies to the software provided by Decawave Ltd. (“Decawave”) in support of its DWM1001 module product (“Module”) all as set out at clause 3 herein (“Decawave Software”).
- (2) Decawave Software is provided in two ways as follows: -
 - (a) pre-loaded onto the Module at time of manufacture by Decawave (“Firmware”);
 - (b) supplied separately by Decawave (“Software Bundle”).
- (3) Decawave Software consists of the following components (a) to (d) inclusive:
 - (a) The **Decawave Positioning and Networking Stack** (“PANS”), available as a library accompanied by source code that allows a level of user customisation. The PANS software is pre-installed and runs on the Module as supplied, and enables mobile “tags”, fixed “anchors” and “gateways” that together deliver the DWM1001 Two-Way-Ranging Real Time Location System (“DRTLS”) Network.
 - (b) The **Decawave DRTLS Manager** which is an Android™ application for configuration of DRTLS nodes (nodes based on the Module) over Bluetooth™.
 - (c) The **Decawave DRTLS Gateway Application** which supplies a gateway function (on a Raspberry Pi ®) routing DRTLS location and sensor data traffic onto an IP based network (e.g. LAN), and consists of the following components:
 - DRTLS Gateway Linux Kernel Module
 - DRTLS Gateway Daemon
 - DRTLS Gateway MQTT Broker
 - DRTLS Gateway Web Manager
 - (d) **Example Host API functions**, also designed to run on a Raspberry Pi, which show how to drive the Module from an external host microprocessor.
- (4) The following third party components are used by Decawave Software and are incorporated in the Firmware or included in the Software Bundle as the case may be: -
 - (a) The PANS software incorporates the Nordic SoftDevice S132-SD-v3 version 3.0.0 (production) which is included in the Firmware and is also included in the Software Bundle;
 - (b) The PANS software uses the eCos RTOS which is included in the Software Bundle. The eCos RTOS is provided under the terms of an open source licence which may be found at: <http://ecos.sourceware.org/license-overview.html>;
 - (c) The PANS software uses an open source CRC-32 function from FreeBSD which is included in the Software Bundle. This CRC-32 function is provided under the terms of the BSD licence which may be found at: <https://github.com/freebsd/freebsd/blob/386ddae58459341ec567604707805814a2128a57/COPYRIGHT>;

- (d) The Decawave DRTLS Manager application uses open source software which is provided as source code in the Software Bundle. This open source software is provided under the terms of the Apache Licence v2.0 which may be found at <http://www.apache.org/licenses/LICENSE-2.0>;
- (e) The Decawave DRTLS Gateway Application uses the following third party components: -
 - (i) The Linux Kernel which is provided as source code in the Software Bundle. The Linux Kernel is provided under the terms of the GPLv2 licence which may be found at: <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html> and as such the DWM1001 driver component of the DRTLS Gateway Application is provided under the same license terms;
 - (ii) The three.js JavaScript library, the downloadable version of which is available here <https://threejs.org/>, is provided under the terms of the MIT Licence which may be found at <https://opensource.org/licenses/MIT>.

Items (a), (b), (c), (d) and (e) in this section 4 are collectively referred to as the “Third Party Software”

- (5) Decawave Software incorporates source code licensed to Decawave by Leaps s.r.o., a supplier to Decawave, which is included in the Firmware and the Software Bundle in binary and/or source code forms as the case may be, under the terms of a license agreement entered into between Decawave and Leaps s.r.o.
- (6) Decawave hereby grants you a free, non-exclusive, non-transferable, worldwide license without the right to sub-license to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating Decawave Software, to modify Decawave Software or incorporate Decawave Software in other software and to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating such modified or incorporated software PROVIDED ALWAYS that the use by you of Third Party Software as supplied by Decawave is subject to the terms and conditions 7()62(i7,)991

SOFTWARE IN WHOLE OR IN PART IN YOUR SYSTEMS AND PRODUCTS RESTS ENTIRELY WITH YOU AND DECAWAVE ACCEPTS NO LIABILITY WHATSOEVER FOR SUCH DECISION.

- (9) DECAWAVE SOFTWARE IS PROVIDED "AS IS". DECAWAVE MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO DECAWAVE SOFTWARE OR USE OF 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.
- (10) 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 DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE. 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.
- (11) 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.
- (12) Decawave reserves the right to make corrections, enhancements, improvements and other changes to its software, including Decawave Software, at any time.

Mailing address: Decawave Ltd.,
Adelaide Chambers,
Peter Street,
Dublin D08 T6YA
IRELAND.

Copyright (c) 15 November 2017 by Decawave Limited. All rights reserved. All trademarks are the property of their respective owners

1 INTRODUCTION

1.1 Overview

The development of real-time positioning network using UWB technology is not a trivial project. The intention of DWM1001 is to simplify the development of UWB RTLS by providing a complete solution in a single module.

The DWM1001 module comes pre-loaded with embedded firmware which provides two-way ranging (TWR) real time location system (RTLS) functionality and networking. The module can be configured and controlled via its API, which can be accessed through a number of different interfaces allowing flexibility to the product designer. The details of the API is described in document [2]. Additionally, Decawave also provide the module firmware in form of libraries and source code along with a build environment so that user can customise the operation and/or add their own functions.

This document describes what is included in the DWM1001 firmware and how these various elements cooperate together, and explains how users can add their own customisations. The aim of this user guide is to help the users with their development based on the DWM1001-DEV board. After reading this guide developers should be able to compile, build and run the DWM1001 firmware, including custom modifications.

1.2 What's included in the firmware release

Each DWM1001 firmware release includes:

- 1) The pre-compiled DWM1001 Firmware, including the firmware Positioning and Networking stack (PANS) library for on-board development, is provided in the DWM1001 on-board package. The firmware and library architecture and its basic partitioning are described in Section 2.
- 2) Firmware tool chain, described in Section 3. The detailed steps to prepare the necessary tool chain, download the firmware user application package, and compile/build/flash their own application are introduced.
- 3) Simple examples demonstrating how to use the DWM1001 APIs. These are described in Section 4.

2 FIRMWARE OVERVIEW

2.1 High-Level Architecture

The firmware embedded in the DWM1001 module basically provides two types of functions: the PANS API and the PANS library which provides lower level functions. The PANS API, includes the Generic API, (these include different API sets for different interfaces and the corresponding parser, which acts as the translator between the user APIs (C, UART, SPI and BLE) and the PANS library). Figure 1 shows the architecture and components of the DWM1001 firmware.

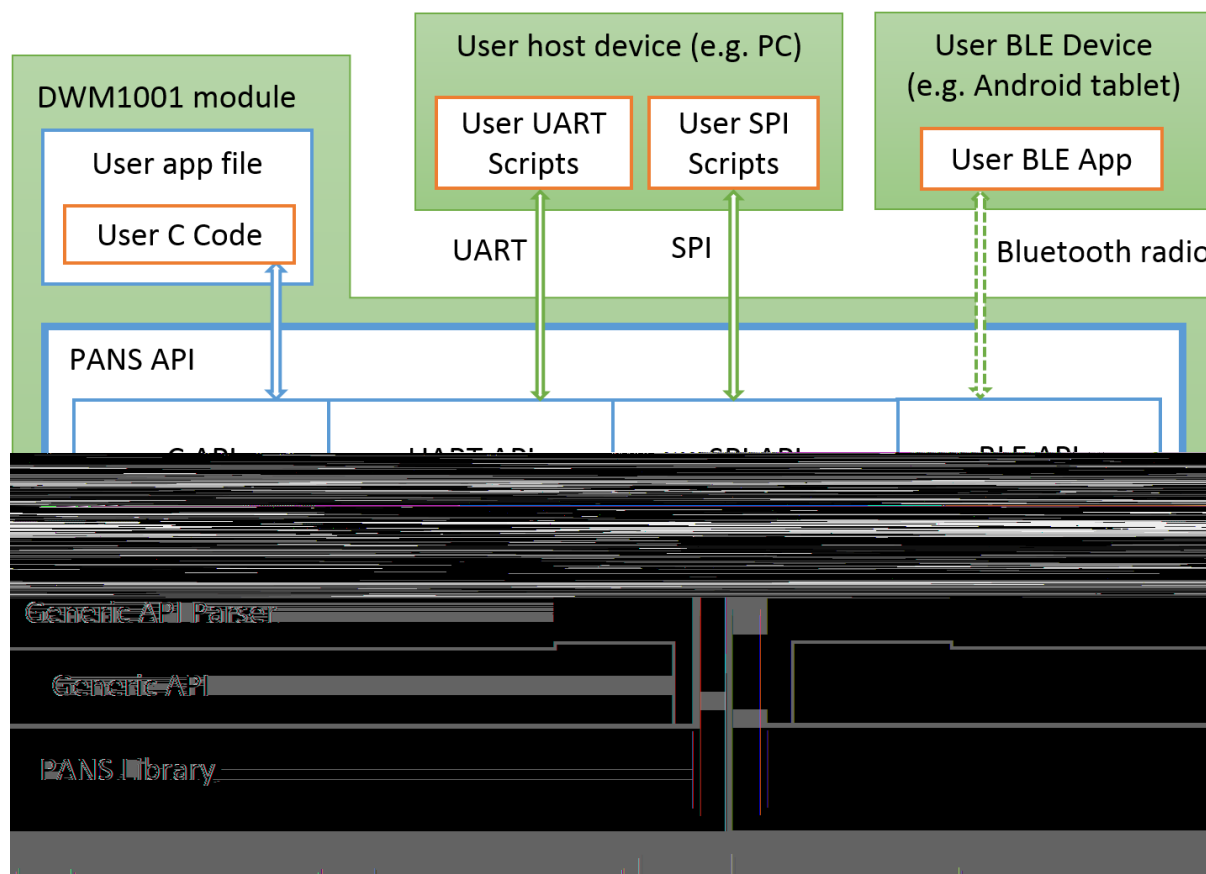


Figure 1 High-Level Architecture of DWM1001 Firmware vs. User Software

As can be seen in Figure 1, apart from using the DWM1001 module itself, the DWM1001 module can be physically connected to external controlling hardware, either wired or wirelessly over Bluetooth radio. The PANS API provides the users with four sets of API to call the PANS library functions through different interfaces:

- **User C code:** an on-board user space, allowing to include an application-specific code in the user application file provided in the DWM1001 firmware, using the firmware development tool chain provided by Decawave, see Section 4.2.
- **SPI:** using a host device (e.g. PC) to communicate with the DWM1001 module using TLV (Type-Length-Value format, detailed in [2]) format requests and responses through SPI bus, for configuration and data transmission.
- **UART:** using a host device (e.g. PC) to communicate the DWM1001 module through UART bus, for configuration and data transmission. Two modes of communications

are provided over the UART interface: UART Generic mode using TLV format requests and responses; and UART Shell mode using terminal prompt commands.

- **BLE:** using a Bluetooth Low Energy (BLE) device (e.g. Android tablet) to control and configure through Bluetooth radio.

All these API sets provide the same set of generic functions calls, namely the Generic API. The Generic API Parser acts as the translator between the four API sets and the Generic API. When an API command is called from any of the above API Interfaces, the command goes through the Generic API Parser which translates the API command into Generic API function calls. If the API command needs a return message, the DWM1001 responds through the same interface.

The use of C code, UART and SPI APIs are detailed with simple examples in Section 4. More detailed information is provided in the API document [2].

Note1: The external interfaces, including the UART, the SPI and the BLE, are used by the external APIs in the PANS library for Host connection. The on-board user application through C code API cannot make use of the external interfaces due to compatibility reasons.

Note2: Decawave do not provide the library source code, or support any use of the PANS library except through the PANS API which is described in the API Document [2]

2.2 Overview of the PANS Library

Figure 2 illustrates the architecture of the PANS Library in detail. From bottom up there are the SoftDevice and the BLE Protocol stack from Nordic Semiconductor, eCos RTOS system with embedded drivers of the components, IoT layer protocols and the applications layer. Section 2.3 gives a brief introduction to each of the components in the library and the operations on the application layer.

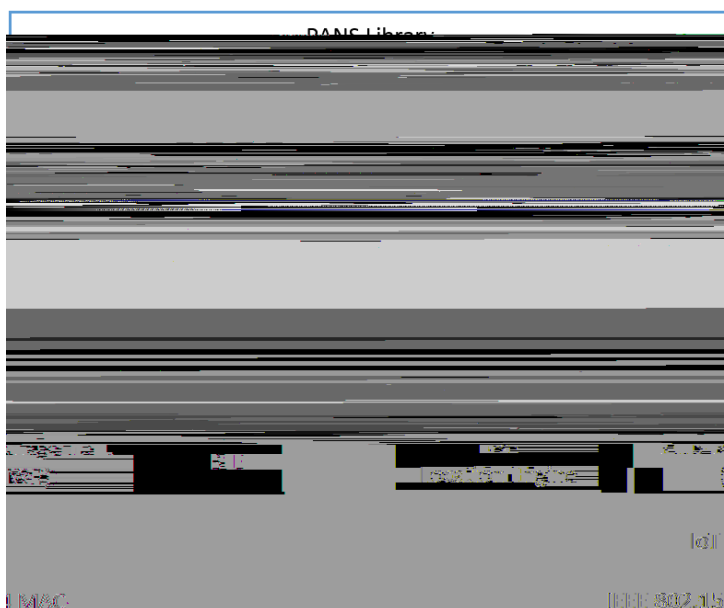


Figure 2 DWM1001 Firmware LibraryComponents

2.3 Components and operations of the PANS Library

The PANS library includes a few firmware components to drive the DWM1001 module.

Some operations are implemented based on these components to perform the positioning and networking function.

2.3.1 SoftDevice and BLE driver

SoftDevice is a Nordic Semiconductor feature-rich library for BLE. The SoftDevice employed on the DWM1001 is the S132, a concurrent multi-link SoftDevice for Central, Peripheral, Broadcaster and Observer roles in BLE applications. The BLE driver / library is included in the S132 SoftDevice, providing the DWM001 with BLE features to create complex network topologies, communication and firmware update over-the-air [1].

2.3.2 eCos RTOS and BSP

eCos RTOS is a free open source real-time operating system. The eCos system provides very good run-time performance and the board support package (BSP) for the DWM1001 hardware platform. It includes all the necessary drivers for the module components (i.e. accelerometer, BLE & DW1000).

2.3.3 DW1000 driver

Decawave's DW1000 API driver. For details please see DW1000 Software API Guide [2].

2.3.4 Stationary indication

An accelerometer component LIS2DH12TR of slave address 0x19 as an I2C peripheral device is implemented on the DWM1001 module, i.e. the read and write address are 0x33 and 0x32 respectively. This accelerometer provides a simple "stationary" indication function. The DWM1001-based tag can be configured to use *Responsive* or *Low-Power* mode. It will

2.3.8 TWR solver / Location Engine

The TWR solver / Location Engine in the tags calculates tag's x, y and z, coordinates is implemented on the DWM1001 module. TWR results between the tag and the relative anchors are sent to the solver for the calculation of tag's position. The TWR results are accessible through the APIs. The internal location engine can be disable and user customised location calculation performed instead, e.g. adding extra filtering of change LE/solver algorithms.

3 FIRMWARE TOOL CHAIN

3.1 Tool Chain Overview

The Decawave firmware tool chain includes the following parts:

- Virtual Box image with necessary tools pre-installed;
- DWM1001 on-board package.

These can be used for developing of a new application with added new functionality which will reside in the DWM1001 module and will run on top of the PANS library and the main module functionality.

3.2 Content in the tool chain

The Figure 3 below shows the tool chain used in the DWM1001 firmware development.

In the DWM1001 product, the whole firmware development is under Linux (Ubuntu) virtual

3.2.1 Hardware part of toolchain

As illustrated in green color in Figure 3, a PC with Microsoft Windows OS and a DWM1001-DEV board are required as the hardware. The DWM1001-DEV board provides the DWM1001 module as the target and a J-Link debugger.

3.2.2 Software part of toolchain

In the DWM1001 firmware development, two types of tools are used:

- 1) Microsoft Windows tools – which need to be downloaded by the users;
- 2) Linux tools – which Decawave have prepared in the Linux VM image.

On the Windows PC, the Virtual Box provided by Oracle needs to be installed. The 32-bit Linux image provided by Decawave needs to be downloaded and imported into the Virtual Box as a VM as introduced in Section 3.3.2.

On the Linux VM, Decawave have prepared the required tools to compile, debug and download the firmware onto the DWM1001 module, including GCC, J-Link and eclipse tools. Decawave would recommend using the pre-installed software tools, since using different versions of the above tools may cause unexpected errors. Decawave do NOT provide any guidance on the installation of the software tools under Linux environment.

3.2.3 Example application package for DWM1001

The DWM1001 on-board package is provided for download on Decawave website [7].

The on-board package includes the DWM1001 PANS library, the Nordic Semiconductor softdevice binary image and the on-board user application example files.

In DWM1001 firmware, the eCos library and other third party software constitute the PANS library as introduced in Section 2.2. These source files are not provided in the on-board package. Folder `dwm\examples\dwm-simple` is the provided on-board user application folder for adding user customized functions through the C code APIs.

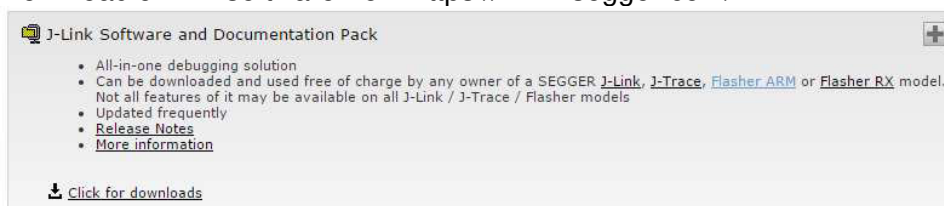
3.3 Guides to flash the DWM1001 with factory image

DWM1001 comes with pre-flashed factory image of firmware. This image is provided in the DWM1001 on-board package: `/dwm/recovery/dwm1001-flash.hex`. Here the guide to flash the factory image through the DWM1001 DEV board is provided.

3.3.1 Using J-Flash Light

The J-Flash Light tool can be used to flash the factory image through the DWM1001 DEV over a few different platforms.

- 1) Prepare j-link tool
 - a. Download J-Link software from <https://www.segger.com/>



- b. Install J-Link software.

- 2) Connect the module with a micro USB data cable, shown in Figure 4.

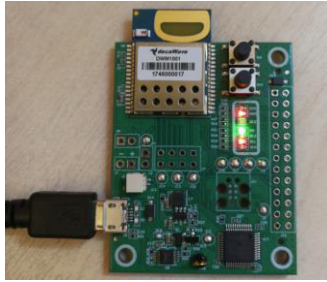

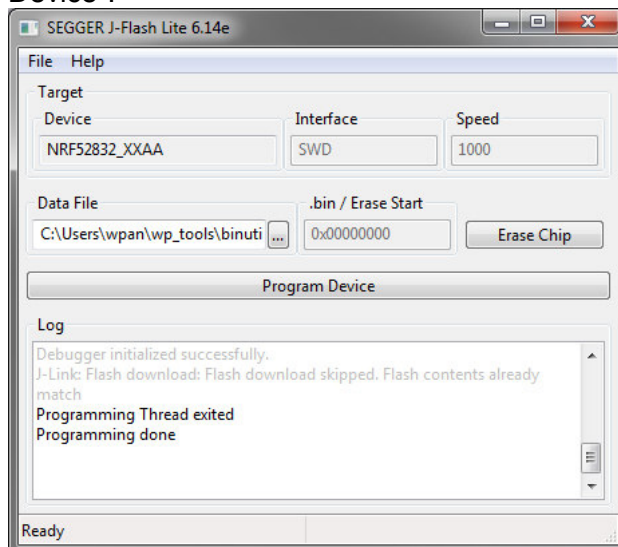


Figure 4 DWM1001 DEV board – micro USB connection

- 3) Flash the image DWM1001 module
 - a. Open J-Flash Lite (default path is similar to in C:\Program Files (x86)\SEGGER\JLink)
 - b. Choose nrf52832_XXAA as Device and SWD as interface, use default speed 1000. Click “OK”.



- c. Click “Erase Chip” to do a full chip erase.
 - d. In Data File, click  and browse to the hex file provided in the DWM1001 on-board package (/dwm/recovery/dwm1001-flash.hex) to flash, click “Program Device”.



The LEDs on the boards should be active once the flash update completes.

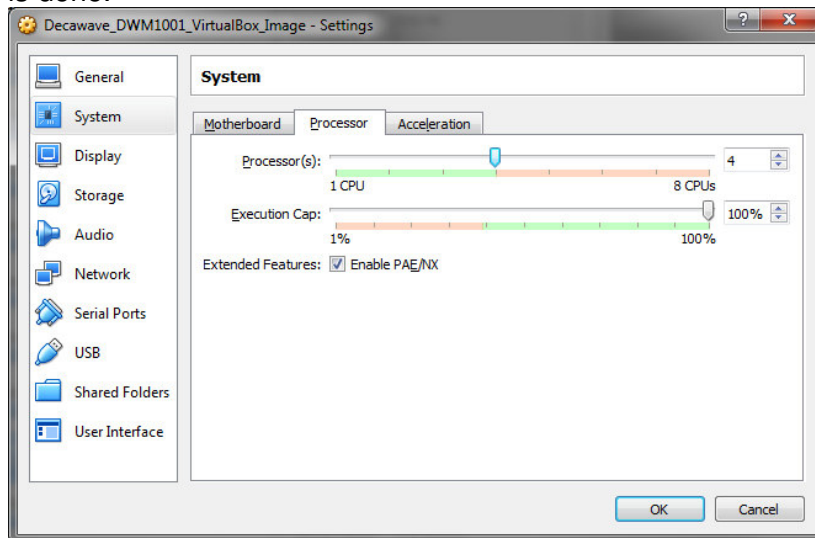
3.3.2 Using the provided VM tools

The provided VM tools can be used to flash the factory image onto a DWM1001 module. The necessary steps are listed here.

- 1) Install Oracle VM VirtualBox [5].
- 2) Download the DWM1001 VM image package [6] and extract.
- 3) Mount the image in VirtualBox Manager. Note that the VM running performance can be improved by increasing the number of processors, RAM and Disk etc.
 - a. Go to the menu bar → Machine → Add...



- b. Find Decawave_DWM1001_VirtualBox_Image.vbox file extracted from the package, and click “Open”.
- c. If a faster speed of the VM is desired, select the created virtual machine “Decawave_DWM1001_VirtualBox_Image”, click “Settings” on the menu bar → System. In Tabs Motherboard and Tab Processor, adjust the suitable amount of memory and processors assigned to the VM. Click “OK” when this is done.



- d. Click “Start” on the menu bar and the virtual machine will start.
 - e. Use password “dw” to login the virtual machine.
- 4) Download the DWM1001 on-board package into ~/Documents. To do this, either downloading directly in the Linux virtual machine or using a shared folder to transmit from the host system (Windows) can work.
- a. In the Linux virtual machine, download the DWM1001 on-board package from [7] into ~/Documents folder using wget command and unzip command.
 - cd ~/Documents
 - wget web/link/to/dwm1001_on-board_package_v1p0.tar
 - tar xf dwm1001_on-board_package_v1p0.tar
 - b. Alternatively setup a shared folder to transmit files between the host Windows system and the Linux virtual machine:
 - i. In Windows, create folder with name “share_folder”, download the DWM1001 on-board package and extract it into this folder.
 - ii. In Virtual box settings, create shared folder using the “share_folder” in Windows. Use auto mount and permanent.
 - iii. Restart the Linux virtual machine, the shared folder will show up in /media/sf_share_folder .
 - iv. In Linux virtual machine, copy the files from /media/sf_share_folder into ~/Documents.

Note: by doing either way above, make sure that the extracted on-board package path looks like ~/Documents/dwm/ .

- 5) Connect the module with a micro USB data cable, see Figure 4 in Section 3.3.1.

- 6) Go to VirtualBox Menu bar → Device → USB → Click SEGGER J-Link
Make sure there is a ✓ before SEGGER J-Link, the device used in DWM1001 DEV board. Having a ✓ in front means the device control is now on the VirtualBox side rather than on the Windows side. See Figure 5.

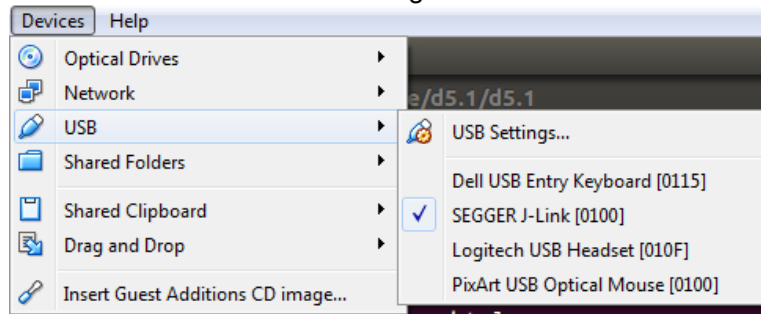


Figure 5 Transfer DWM1001 from Windows to VirtualBox

- 7) Run the script to fully erase the DWM1001 module and flash the module with factory image:
- `cd ~/Documents/dwm/examples/dwm-simple`
 - `make recover`
- The LEDs on the boards should be active once the flash update completes.

4 USER APPLICATION EXAMPLES

4.1 Overview

As illustrated in Figure 1, DWM1001 provides many ways to use its API functions. Examples that show the use of the APIs are listed in this section. In C code, UART Shell, UART Generic, and SPI, examples of getting the location of the node through the API are presented. The API document [2] provides more detailed information.

A tool to open serial port between host device and the DWM1001 module over UART is needed in the firmware development. In Windows, PuTTY can be used; in Linux, minicom can be used. The UART baud rate on the DWM1001 module is 115200 bps.

4.2 “C code API” application example

The VM image [6] in the DWM1001 tool chain has pre-installed Eclipse IDE which will be used for debugging the example below. “C code API” example is an application, which is running as part of the on-board firmware, utilizing a system resources of built-in to the module Cortex M4F microcontroller. The application is running as a thread application in multi-thread environment, driving by included to the PANS library eCos real-time operation system.

4.2.1 Firmware image partitioning

The flash size of the MCU (nRF52832) used on the DWM1001 module is 512KB from address 0 to 0x80000. The partitioning of the flash is illustrated as in Figure 6.

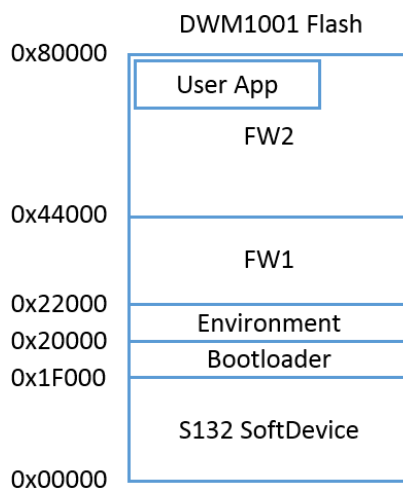


Figure 6 DWM1001 Flash Address Map

The DWM1001 firmware includes the following parts: Nordic S132 Softdevice, Bootloader, Environment, FW1 and FW2:

- **Softdevice**, of size 124KB starting from address 0, provides Bluetooth low energy central and peripheral protocol stack solution.
- **Bootloader**, of size 4KB start from address 0x1F000, is a firmware manager controlling the choice of FW1 and FW2 during booting/reseting.
- **Environment**, of size 8KB start from address 0x20000, is a flash section reserved for the firmware to store user configuration information. Doing power off/on, reset or

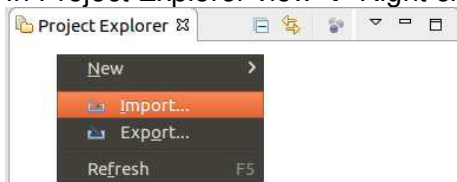
firmware re-flash will not clear the Environment section in the flash. To clear the environment section, a full-erase operation introduced in Section 3.3 is needed. Or alternatively, a shell command “frst” introduced in the PANS API [2] can be used.

- **FW1**, of size 136KB starting from address 0x22000, is a piece of firmware for the over-the-air (OTA) firmware update.
- **FW2**, size of up to 240KB starting from address 0x44000, is the firmware image that includes the full PANS library and the c code user application, where the user application is acting as an independent thread in the firmware and can occupy up to 3KB RAM and 60KB Flash. When rebuilding/reflashing the user application firmware, the whole FW2 is being operated.

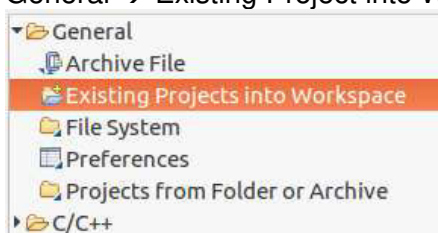
4.2.2 Compiling/debugging user application in the firmware

To add user customized features, the user customized code needs to be added to the application files and the whole project needs to be re-built. A simple example of making a C code user application is given in example/dwm-simple/dwm-simple.c. The detailed steps of setting up the building and compiling with the provided tool chain is listed here.

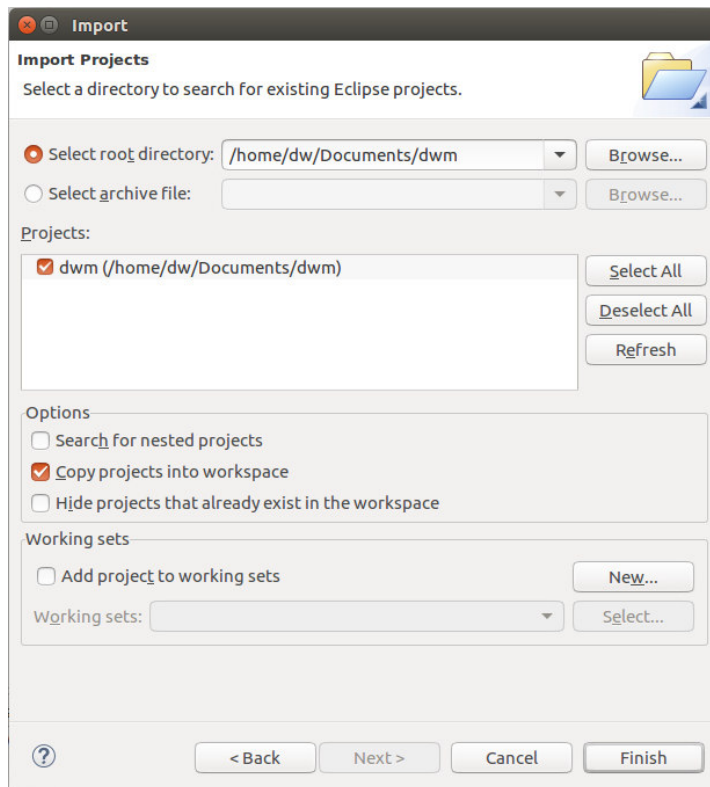
- 1) Install the VM tools and download the DWM1001 on-board package as introduced in Section 3.3.2.
- 2) Open Eclipse
 - a. In the VM, double click Eclipse IDE label on the Desktop
 - b. Alternatively, Eclipse IDE can be found in /data/tools/eclipse folder.
- 3) Import the dwm project:
 - a. In Project Explorer view → Right click → Import...



- b. General → Existing Project into Workspace → Next



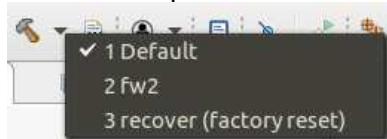
- c. Select root directory → Browse... → find /home/dw/Documents/dwm → OK.
Note: project path may vary according to users' local setup.
- d. Tick the “copy projects into workspace” option so a copy of such project will be created in ~/workspace folder.



e. Click “Finish”.

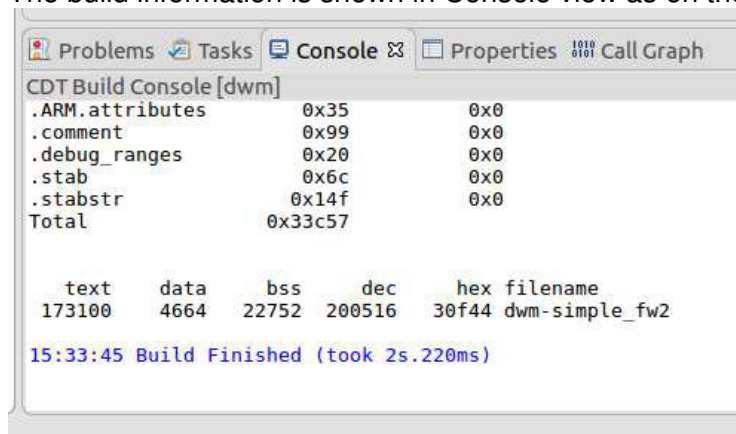
4) Build.

a. Click the dropdown list beside the  button. There are 3 options on the list.



b. Click option “2 fw2” to build the project.

c. The build information is shown in Console view as on the figure below.



d. If the dwm1001 module needs to be flashed with the factory image. Click option “3 recover (factory reset)”. To do this, ensure the device is connected, see step 6) in Section 3.3.2 for more detail.

Note: by doing this, the whole flash including the user configurations will be erased and reset.

5) Add user code.

User customized code can be added in the application project: here an example of using the `dwm_pos_get` API call is given:

In Project Explorer view, expand project dwm, find dwm-simple.c in dwm/examples/dwm-simple/

- a. Find function `app_thread_entry`, add in the local variable area:


```
dwm_pos_t pos;
```
- b. Find `while(1)` in function `app_thread_entry` and add in the brace:


```
dwm_pos_get(&pos);
      printf("x=%ld, y=%ld, z=%ld, qf=%u \n", pos.x, pos.y, pos.z, pos.qf);
      printf("\t\t time=%lu \n", dwm_systime_us_get());
```

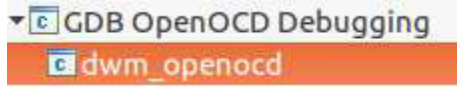
Note: `printf` will send the message to through UART interface when Shell mode is enabled.

- c. Save.
- d. Build the project and eliminate errors.

This change intends to read the position information in the device and print the message on the terminal. A system timer is added in the end of the message to indicate the message time.

The source file `dwm/examples/dwm-simple/dwm-simple.c` can be changed to add/modify functionalities. All available C code API functions are listed in header file `dwm/include/dwm.h`. More detailed information of the C code API is introduced in the DWM1001 API Guide [2].

6) Debug

- a. Connect the DWM1001 device. See step 6) in Section 3.3.2 for more detail.
- b. In Project Explorer view → Right click on dwm project → Debug As → Debug Configurations...
- c. Expand GDB OpenOCD Debugging and choose `dwm_openocd`.
 
- d. Click “Debug”. This will download the firmware onto the DWM1001 module and lead to Debug Perspective.
- e. Now the project can be debugged in Eclipse.

Note1: A few compilation options are provided in the `dwm_user_start()` function, with function names of `dwm_XXXX_compile()`, where XXXX is the component name. Disabling these functions can disable the corresponding components in the DWM1001 firmware.

Note2: It is possible to use breakpoints to debug the firmware. However the user has to be very careful because the nRF52 softdevice interrupts are of highest priority, and if there is BLE activity its interrupts will conflict with user interrupt, thus the BLE should either be disabled or its interrupts masked. Disabling function `dwm_ble_compile()` can disable the BLE compilation so as to void BLE operations during debugging. The function can be re-enabled after debugging.

Note3: SWO debug `printf` is not supported on the DWM1001 DEV board.

4.3 UART applications example

The PANS library provides API functions through UART interface. The connection and simple examples are introduced here.

4.3.1 UART connection

The DWM1001 DEV board provides UART access through both the USB connector and the pins on the external connector. Both accesses are introduced here.

4.3.1.1 UART connection through COM port over USB

The UART connection can be setup simply through a USB data cable as shown in Figure 7. To find the device name of the DWM1001 DEV board in the Windows system:

- 1) Open Device and Printers,
- 2) Find the device J-Link:



Figure 7 J-Link Device in Windows

- 3) Double click the J-Link icon, go to Hardware tab and find the COM port with number as the device name (e.g. in Figure 8 below J-Link is device name is COM28).

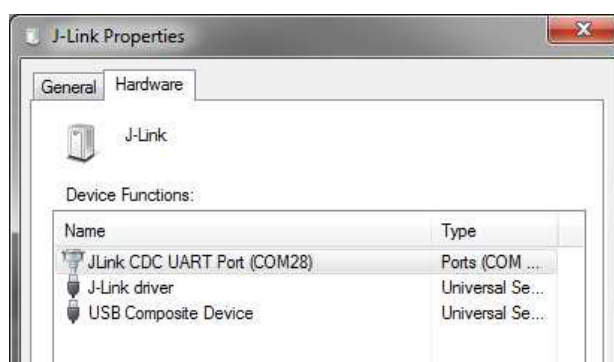


Figure 8 DWM1001 Device COM port over J-Link

In different Linux systems, the UART devices may show different names. In Linux VM, with connection shown in Figure 4 with the control of the device passed to Linux side as introduced in Figure 5, the device name is `/dev/ttyACM0`.

4.3.1.2 UART connection through external connector Tx and Rx pins

Other than using USB cable to connect, the external header pins provided on J10 of the DWM1001 DEV board also provides the UART interface. Table 1 shows the pins needed in the UART connection.

Table 1 UART pin connections

Pin to use in DWM1001 DEV		Pin to be connected to
Connector J10	Function	
Pin 6	GND	GND
Pin 8	RXD	TXD
Pin 10	TXD	RXD

For Raspberry Pi 3 using header pins connection, the device name is `/dev/serial0`.

The connection between the DWM1001 DEV board and a Raspberry Pi 3 (model B) is shown in Figure 9.



Figure 9 Connecting DWM1001 to Raspberry Pi 3 over header pins

Note: Pins on J10 of the DWM1001 DEV board are compatible with Raspberry Pi 3 connector J8 header pins 1-26.

4.3.2 UART examples

UART interface is accessible through two mode, the Shell mode and the Generic mode. Both Generic mode and Shell mode can be used to communicate with the DWM1001 module through the UART connection. The default mode of the DWM1001 UART is Generic mode. The two modes are transferrable:

Generic mode to Shell mode: presses “Enter” twice within one second, or input two bytes [0x0D, 0x0D] within one second.

Shell mode to Generic mode: input “quit” command.

For more information of the two UART API modes, please refer to DWM1001 API Guide [2].

4.3.2.1 UART Shell mode example

With the connection introduced in Figure 4, in VM Linux terminal, open the UART serial port:

```
➤ minicom -D /dev/ttyACM0
```

When seeing “Device or resource busy”, try multiple times until it works.

When seeing “No such file or directory”, check Section 3.3.1 step 6) and Figure 5.

If the connection over UART is successful, “Welcome to minicom” message will show on the terminal. Now hit “Enter” key twice within one second to enter UART shell mode. “dwm>” should present in the terminal when this is all done.

To run the UART Shell command for `dwm_pos_get`, type “apg” followed by “Enter” key. The position of the module in the whole DRTLs will be printed out, see Figure 10. Type “?” followed by “Enter” key to get help information in UART Shell mode. More information of the UART Shell commands is introduced in the DWM1001 API Guide [2].


```

dw@dw:~$ minicom -D /dev/ttyACM0

Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Feb  7 2016, 13:37:32.
Port /dev/ttyACM0, 15:06:24

Press CTRL-A Z for help on special keys

DWM1001 TWR Real Time Location System

Copyright : 2016-2017 LEAPS and Decawave
License   : Please visit https://decawave.com/dwm1001_license
Compiled  : Jul 12 2017 18:00:01

Help      : ? or help

dwm> apg
x:121 y:50 z:251 qf:100
dwm>

```

Figure 10 Connect to DWM1001 device through UART shell

4.3.2.2 UART Generic mode example

A simple example to make use of the UART Generic API is given in the DWM1001 Host Api package [4]. The example runs on the Raspberry Pi platform:

- 1) Download the `dwm1001_host_api` package onto the Raspberry Pi 3 device. Navigate to folder `examples/ex1_TWR_2Hosts/tag/`
- 2) Use nano editor to edit Makefile:
 - nano Makefile
- 3) Change the configuration parameter `USE_INTERFACE` to use UART interface:
 - `USE_INTERFACE = 0`
- 4) Press “Ctrl + o” and “Enter” to save. Press “Ctrl + x” to exit nano editor.
- 5) Use “make” command to build the example:
 - make
- 6) Run the executable:
 - `./tag_cfg`
- 7) Check `log.txt` file for the detail of UART data transmission.

```

pi@rpi-83:~/dwm1001_host_api/examples/ex1_TWR_2Hosts/tag $ ./tag_cfg
dwm_init(): dev0
Opening log file log.txt
  LMH_UARTRX_Init()...
  UART: Init start.
  UART: Init done.
Setting to tag: dev0.
dwm_cfg_tag_set(&cfg_tag): dev0.
Wait 2s for node to reset.
Comparing set vs. get: dev0.

Configuration succeeded.

Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]
Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]
Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]

```

Figure 11 Run simple UART example on Raspberry Pi 3

The simple UART Generic mode example project is designed specifically for Raspberry Pi platform. The source file `examples/ex1_TWR_2Hosts/tag/tag_cfg.c` can be changed to add/modify functionalities. All available UART API functions are listed in header file `include\dwm_api.h`. More detailed information of the UART Generic mode is introduced in the DWM1001 API Guide [2].

The code/makefile needs to be changed, to suit other platforms than Raspberry PI.

4.4 SPI applications example

The DWM1001 module provides APIs over the SPI interface. The connection and a simple example are introduced here.

4.4.1 SPI connection

To connect to the DWM1001 module over SPI, the SPI pins on external connector J10 on the DWM1001 DEV board can be used. Table 2 shows the pins needed in the SPI connection.

Table 2 SPI pin connections

Pin to use		Pin to be connected to
Pin number	Function	
Pin 19	MOSI	MOSI
Pin 21	MISO	MISO
Pin 23	SCLK	SCLK
Pin 25	GND	GND
Pin 24	CSN	CSN

The connection with Raspberry Pi 3 (model B) is shown in Figure 9.

Note: the connector J10 on the DWM1001 DEV board is compatible with Raspberry Pi 3 connector J8 header pins 1-26. Pin 4 from J10 provides 5V power from Raspberry Pi to the DWM1001 DEV board.

4.4.2 SPI example

A simple example to make use of the SPI API is given in the DWM1001 Host API package [4]. The example can run on the Raspberry Pi platform:

- 1) Download the `dwm1001_host_api` package onto the Raspberry Pi 3 device. Navigate to folder `examples/ex1_TWR_2Hosts/tag/`
- 2) Use nano editor to edit Makefile:
 - nano Makefile
- 3) Change the configuration parameter `USE_INTERFACE` to use SPI interface:


```
USE_INTERFACE = 1
```
- 4) Press “Ctrl + o” and “Enter” to save. Press “Ctrl + x” to exit nano editor.
- 5) Use “make” command to build the example:
 - make
- 6) Run the executable:
 - `./tag_cfg`
- 7) Check `log.txt` file for the detail of SPI data transmission.

```
pi@rpi-83:~/dwm1001_host_api/examples/ex1_TWR_2Hosts/tag $ ./tag_cfg
dwm_init(): dev0
Opening log file log.txt
  LMH_SPIRX_Init(): SPI dev0...
  SPI0: spi mode: 0
  SPI0: bits per word: 8
  SPI0: max speed: 8000000 Hz (8000 KHz)
  SPI0: Reseting DWM1001 to SPI:IDLE
  LMH: LMH_SPIRX_Init for SPI dev0 done.
Setting to tag: dev0.
dwm_cfg_tag_set(&cfg_tag): dev0.
Wait 2s for node to reset.
Comparing set vs. get: dev0.

Configuration succeeded.

Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]
Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]
Wait 1000 ms...
dwm_loc_get(&loc):
  [121,50,251,100]
```

Figure 12 Run simple SPI example on Raspberry Pi 3

The simple SPI example project is designed specifically for Raspberry Pi platform. The source file `examples/ex1_TWR_2Hosts/tag/tag_cfg.c` can be changed to add/modify functionalities. All available SPI API functions are listed in header file `include/dwm_api.h`. More detailed information of the SPI API is introduced in the DWM1001 API Guide [2].

The code/makefile needs to be changed to suit other platforms than Raspberry PI.

5 REFERENCES

This document refers to the documents listed below. Note that References' format can be as the author chooses.

1. Nordic nRF52 series Softdevice introduction, available from www.nordicsemi.com
2. DW1000 Software API Guide, available from www.decawave.com
3. DWM1001 System Overview, available from www.decawave.com
4. DWM1001 Host API source code package available from www.decawave.com
5. Oracle VM VirtualBox, available from www.virtualbox.org
6. DWM1001 VM image, available from www.decawave.com
7. DWM1001 on-board source code package, available from www.decawave.com
8. DWM1001 Android application source package, available from www.decawave.com

6 DOCUMENT HISTORY

6.1 Revision History

Table 3: Document History

Revision	Date	Description
1.0	18-Dec-2017	Initial version.
1.1	11-Jan-2018	Editorial changes

7 CHANGE LOG

Revision 1.1

Page	Change Description
27	Remove unreleased documents from references
27	Repair url links for 2 references
10	Remove paragraph related to fig 2
10	Renumber references

8 ABOUT DECAWAVE

Decawave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4-2011 UWB standard. This device is the first in a family of parts that will operate at data rates of 110 kbps, 850 kbps and 6.8 Mbps.

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

Further Information

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

Decawave Ltd
Adelaide Chambers
Peter Street
Dublin 8
D08 T6YA
t: +353 1 6975030
e: sales@decawave.com
w: www.decawave.com

DWM1001 System Overview And Performance

Version 1.0

**This document is subject to change without
notice**

TABLE OF CONTENTS

1	INTRODUCTION	9
1.1	OVERVIEW	9
1.2	AUDIENCE.....	9
1.3	THE DWM1000 MODULE AND RTLS	9
1.4	IMPORTANT NOTICE ON RELEASES	9
1.5	MORE INFORMATION	9
2	SUMMARY PERFORMANCE	11
3	TARGET SYSTEM ARCHITECTURE	12
3.1	SCHEDULE NOTE: RELEASE 1 VS. RELEASE 2.....	12
3.2	DRTLS NETWORK.....	13
3.3	DWM1001 TWO-WAY-RANGING REAL TIME LOCATION SYSTEM (DRTLS).....	13
3.3.1	<i>Schedule Note: Release 1 vs. Release 2</i>	13
3.4	GATEWAY	13
3.5	END POINTS.....	14
3.5.1	<i>DWM1001 Web Clients</i>	14
3.5.2	<i>MQTT Clients</i>	14
3.5.3	<i>Local Bluetooth-connected Tablets/Smartphones</i>	15
4	DRTLS NETWORK CONFIGURATION AND CONTROL	16
4.1	DRTLS OPERATION	16
4.2	NETWORK INITIALISATION.....	17
4.2.1	<i>Criteria for anchor sign-up</i>	17
4.2.2	<i>Sign-up process</i>	17
4.3	INFRASTRUCTURE COLLISION DETECTION AND RESOLUTION.....	18
4.4	OPERATION OF A CONNECTED ANCHOR/ANCHOR-INITIATOR	18
4.5	OPERATION OF A TAG	19
4.6	TWR PROTOCOL AND TWR SLOT RESERVATION	20
4.7	TWR COLLISION DETECTION AND RESOLUTION	21
4.8	TAG'S TWR STRATEGY.....	21
5	LOCATION ENGINE	23
6	POWER MANAGEMENT STRATEGY	24
6.1	POWER MANAGEMENT CONTROL OF MAIN SYSTEM COMPONENTS	24
6.2	WAKE-UP SOURCES	25
6.3	TWO LOCATION UPDATE RATES.....	25
7	SCALABILITY	26
7.1	SYSTEM EXPANSION - ANCHORS	26
7.1.1	<i>Scaling Rules</i>	27
7.2	INSTALLATION LIMITATIONS	27
7.2.1	<i>Limitation 1: Maximum number of anchor seats is 16</i>	27
7.2.2	<i>Limitation 2: Maximum number of clock level is 127</i>	27
7.3	SYSTEM EXPANSION – TAGS.....	28
7.4	NETWORK COVERAGE AND EXPANSION.....	28
8	MEMORY USAGE AND FIRMWARE UPDATE	30
8.1	FIRMWARE UPDATE.....	30
8.2	FIRMWARE UPDATE INITIATED OVER BLUETOOTH	30

8.3	FIRMWARE UPDATE VIA UWB.....	30
8.4	MANUAL FIRMWARE UPDATE.....	31
9	APPENDIX: FRAME FORMATS	32
9.1	IEEE 802.15.4 FRAME	32
9.1.1	Beacon message.....	33
9.1.2	Join request message	34
9.1.3	Join confirmation message.....	35
9.1.4	Almanac message	35
9.1.5	Service message	36
9.1.6	Firmware Update Data Request message.....	36
9.1.7	Firmware Update Data message.....	36
9.1.8	Position message.....	37
9.1.9	Group Poll message.....	37
9.1.10	Poll message.....	38
9.1.11	Response message	38
9.1.12	Final message.....	38
10	REFERENCES	40
10.1	LISTING.....	40
11	DOCUMENT HISTORY	41
11.1	REVISION HISTORY.....	41
11.2	MAJOR CHANGES	41
12	ABOUT DECAWAVE	42

LIST OF TABLES

TABLE 1:	SUMMARY SYSTEM PERFORMANCE	11
TABLE 2:	IEEE 802.15.4 DATA FRAME STRUCTURE.....	32
TABLE 3:	IEEE 802.15.4 ACK FRAME	33
TABLE 4:	TABLE OF REFERENCES.....	40
TABLE 5:	DOCUMENT HISTORY	41

LIST OF FIGURES

FIGURE 1:	TARGET SYSTEM ARCHITECTURE	12
FIGURE 2:	DRTLS NETWORK.....	13
FIGURE 3:	GATEWAY COMPONENTS	14
FIGURE 4:	WEB-CLIENT INTERFACE MOCK-UP	14
FIGURE 5:	LOCAL CONFIGURATION & VISUALISATION ON THE ANDROID APPLICATION	15
FIGURE 6:	SUPERFRAME STRUCTURE.....	16
FIGURE 7:	TAG OPERATION IN LOW-POWER MODE VS RESPONSIVE MODE	19
FIGURE 8:	SUPERFRAME SHOWING TWR FRAMES.....	20
FIGURE 9:	TAGS CHOICE OF ANCHORS FOR TWR.....	22
FIGURE 10:	MAIN DWM1001 SYSTEM COMPONENTS.....	24
FIGURE 11:	SCALING THE DRTLS SHOWING ANCHOR'S SEAT NUMBERS	26
FIGURE 12:	SCALING THE DRTLS SHOWING CLOCK LEVELS	27
FIGURE 13:	SINGLE GATEWAY AREA COVERAGE WITH STAR TOPOLOGY	29
FIGURE 14:	SINGLE GATEWAY AREA COVERAGE WITH LINE TOPOLOGY	29

FIGURE 15: DWM1001 FLASH ADDRESS MAP 30
FIGURE 16: IEEE 802.15.4 MAC FRAME FORMAT, WITH E.G. BROADCAST ADDRESS AND 33

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 with Decawave for the most recent updates on this product.

Copyright © 2017 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

- (1) This Disclaimer applies to the software provided by Decawave Ltd. (“Decawave”) in support of its DWM1001 module product (“Module”) all as set out at clause 3 herein (“Decawave Software”).
- (2) Decawave Software is provided in two ways as follows: -
 - (a) pre-loaded onto the Module at time of manufacture by Decawave (“Firmware”);
 - (b) supplied separately by Decawave (“Software Bundle”).
- (3) Decawave Software consists of the following components (a) to (d) inclusive:
 - (a) The **Decawave Positioning and Networking Stack** (“PANS”), available as a library accompanied by source code that allows a level of user customisation. The PANS software is pre-installed and runs on the Module as supplied, and enables mobile “tags”, fixed “anchors” and “gateways” that together deliver the DWM1001 Two-Way-Ranging Real Time Location System (“DRTLS”) Network.
 - (b) The **Decawave DRTLS Manager** which is an Android™ application for configuration of DRTLS nodes (nodes based on the Module) over Bluetooth™.
 - (c) The **Decawave DRTLS Gateway Application** which supplies a gateway function (on a Raspberry Pi ®) routing DRTLS location and sensor data traffic onto an IP based network (e.g. LAN), and consists of the following components:
 - DRTLS Gateway Linux Kernel Module
 - DRTLS Gateway Daemon
 - DRTLS Gateway MQTT Broker
 - DRTLS Gateway Web Manager
 - (d) **Example Host API functions**, also designed to run on a Raspberry Pi, which show how to drive the Module from an external host microprocessor.
- (4) The following third party components are used by Decawave Software and are incorporated in the Firmware or included in the Software Bundle as the case may be: -
 - (a) The PANS software incorporates the Nordic SoftDevice S132-SD-v3 version 3.0.0 (production) which is included in the Firmware and is also included in the Software Bundle;
 - (b) The PANS software uses the eCos RTOS which is included in the Software Bundle. The eCos RTOS is provided under the terms of an open source licence which may be found at: <http://ecos.sourceware.org/license-overview.html>;
 - (c) The PANS software uses an open source CRC-32 function from FreeBSD which is included in the Software Bundle. This CRC-32 function is provided under the terms of the BSD licence which may be found at: <https://github.com/freebsd/freebsd/blob/386ddae58459341ec567604707805814a2128a57/COPYRIGHT>;

- (d) The Decawave DRTLS Manager application uses open source software which is provided as source code in the Software Bundle. This open source software is provided under the terms of the Apache Licence v2.0 which may be found at <http://www.apache.org/licenses/LICENSE-2.0>;
- (e) The Decawave DRTLS Gateway Application uses the following third party components: -
 - (i) The Linux Kernel which is provided as source code in the Software Bundle. The Linux Kernel is provided under the terms of the GPLv2 licence which may be found at: <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html> and as such the DWM1001 driver component of the DRTLS Gateway Application is provided under the same license terms;
 - (ii) The three.js JavaScript library, the downloadable version of which is available here <https://threejs.org/>, is provided under the terms of the MIT Licence which may be found at <https://opensource.org/licenses/MIT>.

Items (a), (b), (c), (d) and (e) in this section 4 are collectively referred to as the “Third Party Software”

- (5) Decawave Software incorporates source code licensed to Decawave by Leaps s.r.o., a supplier to Decawave, which is included in the Firmware and the Software Bundle in binary and/or source code forms as the case may be, under the terms of a license agreement entered into between Decawave and Leaps s.r.o.
- (6) Decawave hereby grants you a free, non-exclusive, non-transferable, worldwide license without the right to sub-license to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating Decawave Software, to modify Decawave Software or incorporate Decawave Software in other software and to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating such modified or incorporated software PROVIDED ALWAYS that the use by you of Third Party Software as supplied by Decawave is subject to the terms and conditions of the respective license agreements as set out at clause 4 herein AND PROVIDED ALWAYS that Decawave Software is used only in systems and products based on Decawave semiconductor products. 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.
- (7) Downloading, accepting delivery of or using Decawave Software indicates your agreement to the terms of (i) the license granted at clause 6 herein, (ii) the terms of this Disclaimer and (iii) the terms attaching to the Third Party Software. If you do not agree with all of these terms do not download, accept delivery of or use Decawave Software.
- (8) 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 AND DECAWAVE ACCEPTS NO LIABILITY WHATSOEVER FOR SUCH DECISION.

- (9) DECAWAVE SOFTWARE IS PROVIDED "AS IS". DECAWAVE MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO DECAWAVE SOFTWARE OR USE OF 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.
- (10) 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 DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE. 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.
- (11) 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.
- (12) Decawave reserves the right to make corrections, enhancements, improvements and other changes to its software, including Decawave Software, at any time.

Mailing address: Decawave Ltd.,
Adelaide Chambers,
Peter Street,
Dublin D08 T6YA
IRELAND.

Copyright (c) 15 November 2017 by Decawave Limited. All rights reserved. All trademarks are the property of their respective owners

1 INTRODUCTION

1.1 Overview

This document describes the details and performance of DWM1001 Two-Way-Ranging Real Time Location System (DRTL5). The major components are:

- Decawave DWM1001 module hardware
- Decawave Positioning and Networking Stack (PANS) software

1.2 Audience

This document is suitable for customers (engineers, RTLS system designers) of the DWM1001 [1], DWM1001-DEV [2] and MDEK1001 [3] products who are integrating the technology into their own products.

1.3 The DWM1000 module and RTLS

The DWM1001 is a module product that comes complete with firmware to allow system developers to quickly implement an RTLS to suit their particular end application, or add RTLS capability to an existing system. The module may be configured to behave as an “anchor” one of the fixed nodes in the system or a “tag” one of the mobile located nodes in the system. The module configuration may be achieved either via Bluetooth using the companion application (Decawave DRTL5 Manager [4]); via an SPI or UART connection from an external host [5] or remotely from a Web client via the UWB backhaul.

The module incorporates Decawave’s DW1000 UWB transceiver which the module’s on-board firmware drives to implement the network of anchor nodes and perform the two-way ranging exchanges with the tag nodes enabling each tag to compute its own location.

The module also incorporates the Nordic Semiconductor nRF52832 IC providing the Bluetooth connectivity used for configuration and the microprocessor that runs the firmware which drives the DW1000 and provides the RTLS enabling functionality. A more complete description of this may be found in section 3.

The module is typically mounted on a PCB, such as the DWM1001-DEV product. The MDEK1001 kit provides 12 DWM1001 modules already mounted on “development” boards enabling system developers evaluate the product and/or begin their system development before embarking on their own designs.

1.4 Important Notice on Releases

The DWM1001 module and DWM1001-DEV and MDEK1001 products will be launched using a “Release 1” version of the firmware and accompanying tablet software (Decawave DRTL5 Manager). This will be limited to configuration and visualisation of the network via the Decawave DRTL5 Manager on a tablet that is in-range of the tags, or direct connection of those tags to a PC, or collection of tag information via a listener device to the PC.

Subsequently, a “Release 2” version of the firmware will include data networking, security and interaction with a gateway device to facilitate building a centralised RTLS network.

1.5 More Information

Information about the DWM1001 and related products can be found in the following documentation:

- DWM1001 (module)
 - DWM1001 Product Brief
 - DWM1001 Hardware Datasheet
 - DWM1001 Firmware User Guide
 - DWM1001 Firmware API Guide
 - DWM1001 Bluetooth API Guide
- DWM1001-DEV (development board)
 - DWM1001-DEV Product Brief
 - DWM1001-DEV Hardware Datasheet
 - DWM1001-DEV Quick Start Guide (in the box)
- MDEK1001 (development and evaluation kit)
 - MDEK1001 Product Brief
 - MDEK1001 System User Manual
 - MDEK1001 Quick Start Guide (in the box)
 - MDEK1001 Application Manager Source Code Guide
- www.decawave.com

2 SUMMARY PERFORMANCE

The table below summarises the performance of the DRTLs. See the DWM1001 datasheet for more detailed information about the module hardware.

Table 1: Summary System Performance

Parameter	Description	Notes
RTLS System Performance		
X-Y location accuracy	<10 cm (typical)	Line-of-Sight (LOS)
UWB Range (node to node LOS)	~60 m	
System capacity / cluster	150 Hz	750 tags @ 0.2 Hz 150 tags @ 1 Hz 15 tags @ 10 Hz etc.
Max. Location Rate / Tag	10 Hz	
Min. Location Rate / Tag	0.0167 Hz	Every 1 minute
Max # Anchors (theoretical)	Area Dependent	See section 7
Max. # Tags / cluster (theoretical)	9000	@ min. rate of 0.0167 Hz (every 1 minute)
Tag Power Consumption (ranging)	TBD	
Tag Power Consumption (sleep)	<5 μ A	
Anchor Power Consumption (ranging)	TBD	
Tag Battery Life (estimates)	TBD	
Available Memory		
Flash Memory available to user	Release 1: 60 kB Release 2: TBD	
RAM available to user	Release 1: 3 kB Release 2: TBD	
Data Throughput		
Throughput over backhaul	Uplink: 255 B location data 640 B sensor data Downlink: 240 B sensor data	Each superframe (100 ms) Release 2 (see section 1.4)
Hops	3	Max Release 2 (see section 1.4)
System Latency	100 ms per hop	From tag to gateway Release 2 (see section 1.4)
UWB Parameters		
UWB Channel	Channel 5 (6.5 GHz)	Fixed
Data Rate	6.81 Mbps	Fixed
PRF	64 MHz	Fixed
Preamble Length	128	Fixed
Preamble Code	9	Fixed

3 TARGET SYSTEM ARCHITECTURE

The DWM1001 hardware and software is designed to enable users to quickly build a system like that shown in Figure 1.

The main components of such a system are:

- DWM1001 Two-Way-Ranging Real Time Location System (DRTL) Network:
 - A UWB network consisting of anchors and tags based on the DWM1001 module.
 - A gateway to connect the anchors and tags UWB network to the broader IP network. This consists of a DWM1001 configured as a bridge node and a Linux host (e.g. Raspberry PI)
 - Local configuration and visualisation via an Android application
- Backhaul infrastructure e.g. Ethernet or Wi-Fi
- End Points (consumer of data):
 - DWM1001 Web Clients
 - MQTT Clients
 - Local Bluetooth-connected Tablets/Smartphones

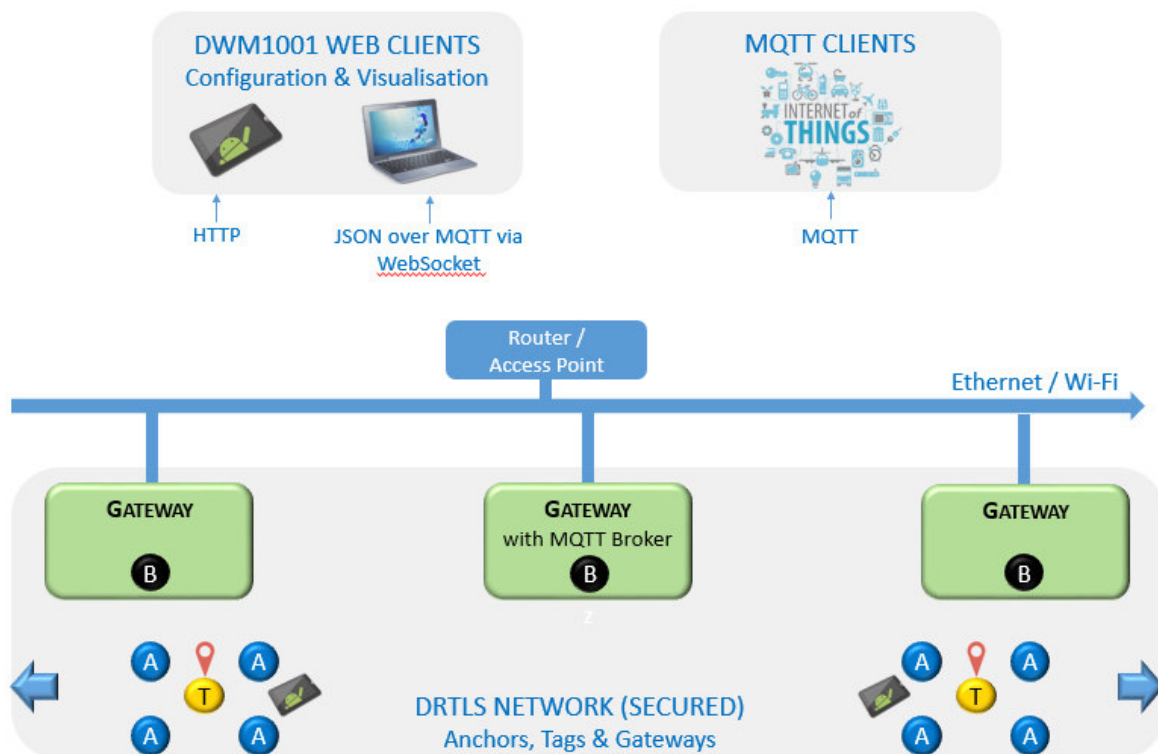


Figure 1: Target System Architecture

3.1 Schedule Note: Release 1 vs. Release 2

The components to build the system in Figure 1 will be delivered across “Release 1” and “Release 2” firmware.

“Release 1” will focus on the DRTL network layer (without the gateway functionality). See Section 3.2 for more details.

3.2 DRTLS Network

3.3 DWM1001 Two-Way-Ranging Real Time Location System (DRTLS)

The DRTLS network components are shown in Figure 2. The DRTLS consists of:

- A number of DWM1001 devices which can be configured as an anchor, a tag or a bridge node on a gateway. The tags are usually mobile, and anchors fixed in place.
- An optional gateway unit, containing a DWM1001 module, which connects the system to the outside network (LAN/WAN).
- External applications (PC/server and tablet/phone – e.g. Decawave DRTLS Manager) which can be used to install, commission and configure the DRTLS. The tag's location can then be displayed at either the local application (tablet) or remote application.

The operation of the DRTLS network is described in Section 4.

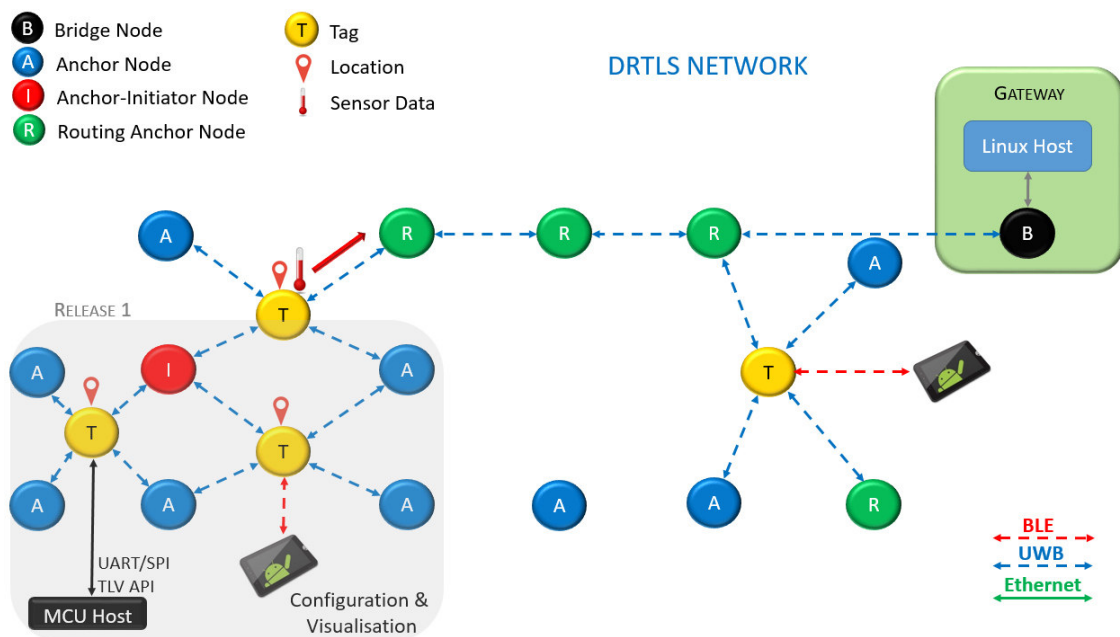


Figure 2: DRTLS Network

3.3.1 Schedule Note: Release 1 vs. Release 2

The components to build the system in Figure 1 will be delivered across “Release 1” and “Release 2” firmware.

“Release 1” firmware will deliver the items in the shaded box of Figure 2.

3.4 Gateway

The gateway components are shown in Figure 3.

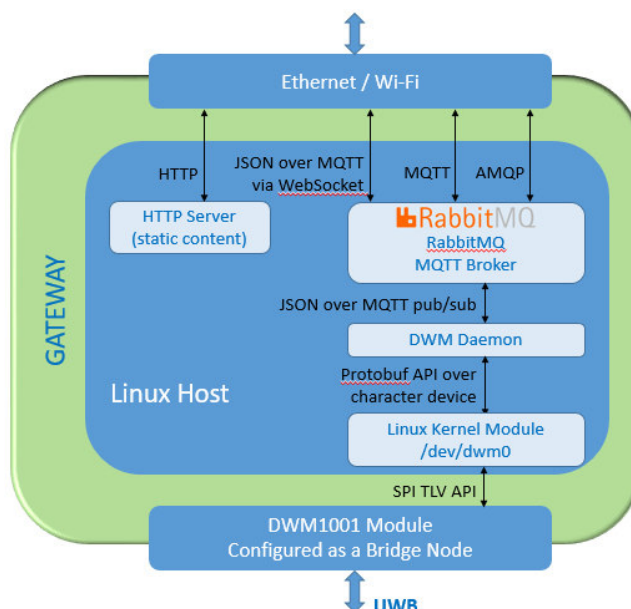


Figure 3: Gateway Components

3.5 End Points

The data from the system is visualised via Web, MQTT or Bluetooth clients.

3.5.1 DWM1001 Web Clients

Web-based clients on a PC or Tablet will provide the interface for configuration and visualisation of the location data. Figure 4 shows a mock-up of this interface.



Figure 4: Web-Client Interface Mock-Up

3.5.2 MQTT Clients

MQTT clients will be able to connect to the MQTT Broker to access the DWM1001 data. This will be specified further as part of Release 2 delivery.

3.5.3 Local Bluetooth-connected Tablets/Smartphones

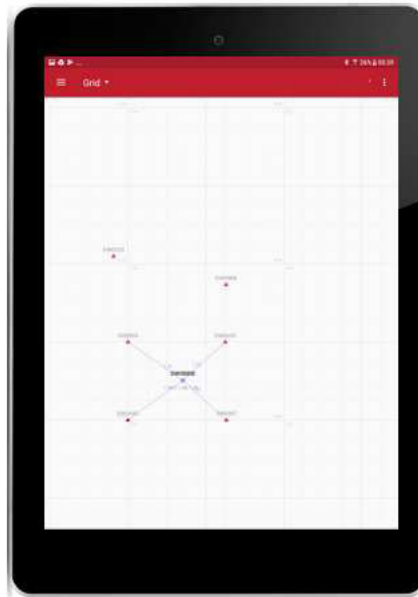


Figure 5: Local Configuration & Visualisation on the Android Application

4 DRTLS NETWORK CONFIGURATION AND CONTROL

4.1 DRTLS operation

A UWB DRTLS system/network uses a minimum of three anchors to locate a tag. To initialise the system at least one of the anchors must be configured as an “initiator”. The initiator anchor will start and control the network and allow other anchors to join and form a network.

The DRTLS uses TDMA channel access. The nodes operate using a repeating “superframe” structure of 100 ms duration. This structure is shown in Figure 6. The initiator controls the timing and the superframe starts with 16 Beacon Message (BCN) slots, in which anchors send *Beacon* messages (0), this is followed by two Service (SVC) slots which are used for *Almanac* (9.1.4) and network service messages (e.g. network join request). There are also 15 Two-Way-Ranging (TWR) slots which are used for tag to anchor two-way ranging exchanges. There is some Idle time reserved at the end of the superframe for addition of backhaul data traffic (will be added in Release 2).

The DRTLS uses a TWR scheme, in which tag ranges with up to four anchors, and then calculates its own location with respect to the anchors’ positions, which it has learnt from the *Beacon* messages. The tags will initially listen for *Beacon* and *Almanac* messages and learn about the network topology, and selects four anchors to range with. The details of ranging are given in 4.6. The tag’s position is then sent via Bluetooth to the Decawave DRTLS Manager application (when tag is in *Responsive* mode, see section 4.5), where it can be visualised. Note, the location and ranging information can also be output over UART, when a tag is connected to a host device or a PC. This is further described in [5].

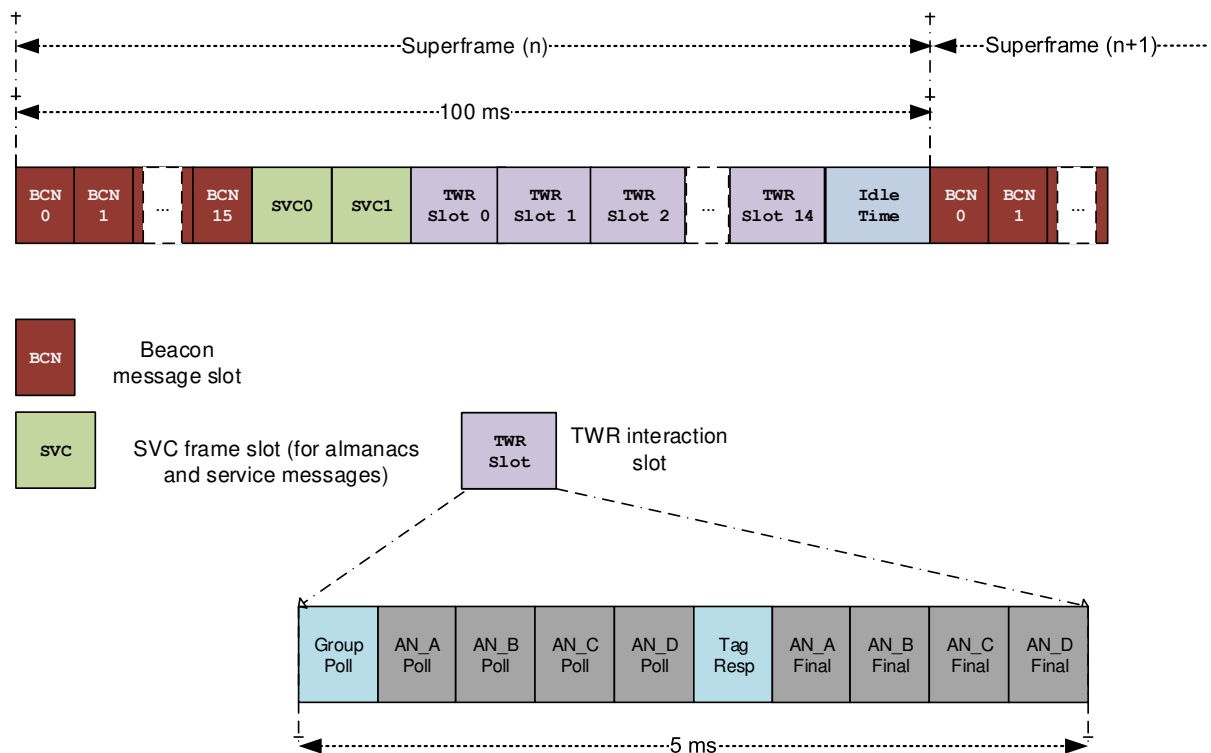


Figure 6: Superframe structure

4.2 Network initialisation

It is possible to configure more than one anchor as an initiator, but only one will be active and the others will take a role of ordinary anchor. Every initiator begins by listening for UWB messages for a period, and only when it receives none does it act as an initiator to initialise the network by choosing seat number 0, meaning that it will transmit its *Beacon* messages in BCN slot 0, and it will start a DRTLS network by broadcasting *Beacons* and *Almanacs*.

When an anchor begins its operation it listens for beacons from other anchors and tries to join the network. Once they join the DRTLS network they will also send *Beacon* and *Almanac* messages. Which contain the anchor's coordinates, general network information and anchor's TWR involvement with tags.

Each anchor uses one BCN slot and since there are 16 BCN slots in the superframe this means that at most 16 anchors can operate in the same area, each occupying its own BCN time slot.

Note: This does not mean that the system is limited to 16 anchors, but rather that across the network no new anchor can be added (take a BCN slot) if any of its in-range anchors can hear another anchor already occupying that slot. This mechanism is described in detail below, and further in section 7.1

4.2.1 Criteria for anchor sign-up

Before an anchor can join an existing network, the following criteria must be satisfied:

- The cluster map and cluster neighbor map (sent as part of *Beacon* messages) must indicate there is a free seat to occupy. (i.e. the number of occupied BCN slots in the superframe is < 16)
- Clock level of the in-range anchors is lower than 127. The anchors which can hear initiator's *Beacons* are at clock level 1. The next ones are clock level 2, etc.
- All in-range anchors have confirmed that the requested seat (requested by the anchor wishing to join) is free and no collisions with the other devices occurred during the sign-up process.
- The firmware must be compatible (if FW update is enabled), i.e. the version must match the version in the initiator. The firmware version is sent in the *Almanac* messages.

4.2.2 Sign-up process

An anchor firstly listens for the *Almanac* messages and checks its hardware version, firmware version and firmware size are compatible with the other devices on the network. If the version is different, then a firmware update process is initiated, see section 8, otherwise the anchor can continue the sign-up process. The over-the-air (OTA) firmware update is enabled by default in DRTLS. Note: the initiator firmware can be updated by SWD or BLE. After that each node will be updated by initiator over UWB.

The new anchor continues listening to the *Beacons* and creates a list of networked anchors which it can hear. After it hears each of the anchors at least 3 times, and sees that there is a free seat, it starts looking for a free SVC slot to send a *Cluster Join Request* message. The networked anchors indicate in their *Beacons* when the SVC slot can be used for Up-Link, i.e. is available for joining anchor to send the *Cluster Join Request* message to the infrastructure.

The *Cluster Join Request* message contains information about the joining anchor (hardware

version, firmware version, firmware checksum, and other node capabilities) and the requested seat number. The joined anchors send a response to the request using the extended part of the *Beacon* message. They embed a *Cluster Join Confirmation* message at the end of the *Beacon* messages. They repeat this for 3 cycles (superframes) since the last reception of the *Cluster Join Request* message. During this exchange the anchors in the network and the joining anchor are “locked” and no new anchors will be able to join. If any new anchor sends a join request it will be ignored, this means only one anchor can join at a time. The first *Cluster Join Request* message will contain seat number 0xFF which means, the joining anchor is probing if the networked anchors already have it on their lists.

These scenarios can happen:

- All networked anchors which the joining anchor can hear will respond with a valid seat number (0 to 15), that means the joining anchor was connected to the network recently, and is reconnecting. It would not need to be assigned a new seat and can continue to use the previous seat if all networked anchors replied with the same number. If the replied seat numbers are not the same, then the joining anchor will need to choose a free seat (i.e. one which none of the networked anchors are using). If no free seat is available the new anchor cannot join the network, and will have to wait and try to connect later.
- Some networked anchors will respond with a valid seat number and some with seat number 0xFF. The seat number 0xFF means those anchors don't know about the joining anchor yet. The joining anchor will need to choose a free seat (i.e. one which none of the networked anchors are using).
- Some networked anchors don't respond after the *Cluster Join Request* message. The joining anchor will mark these as gone (i.e. out of range). When all the other anchors are locked to it, it can choose a free seat.

Once the joining anchor has chosen a seat it will send *Cluster Join Request* message with the requested seat number and then when all networked anchors confirm the seat, it will consider itself connected and the sign-up process has completed successfully. The anchor will then start sending the *Beacon* and *Almanac* messages, and participating in the DRTLS.

4.3 Infrastructure collision detection and resolution

An *infrastructure collision* occurs when two anchors are transmitting in the same BCN slot (i.e. occupying the same seat). Each anchor participating in the network, receives *Beacon* messages from all other anchors that are in range, and at the same time monitors for any collisions. Each anchor then maintains a list of anchors for which it has detected collisions/conflicts, i.e. it detects that the same seat is used by two different anchors, it will report this if the collision counter reaches a threshold during a defined period. The anchor receiving a collision report addressed to it will leave the network and attempt to re-join which should result in it occupying free seat number.

The collision is reported using a SVC slot in the *Service* message with parameters indicating the address of the colliding anchor and the address of the reporting anchor.

4.4 Operation of a connected anchor/anchor-initiator

An anchor which can hear the initiator keeps its clock synchronised with it, so that it is aligned to the initiator's superframe timings. This anchor's clock level is said to be 1. An anchor which cannot hear the initiator will keep its clock synchronised with the anchor which is closer to the initiator (e.g. if it can hear two anchors, one with level 1 and other with level 3, it will use level 1 anchor's clock for its clock synchronisation estimation). The further the anchor is from the initiator the greater its clock level will be. The DRTLS supports a

maximum clock level of 127.

Each anchor will send *Beacons* in its reserved BCN slot, based on its seat number. And also send *Almanacs* in the SVC slot during its reserved superframe. Each anchor, based on its seat number, has a reserved SVC slot in which to send the Almanac. The *Beacons* are sent every superframe and *Almanacs* every 32nd superframe.

The anchor will listen during other BCN slots for any *Beacons* and during SVC slots for any *Almanacs* or other *Service* messages. It will listen at the start of the TWR slot for *Group Poll* messages. If the *Group Poll* contained its address, then it will respond with a *Poll* message and perform TWR with the tag.

Upon reception of *Firmware Update Data Request* message the anchor will provide firmware to requesting device over UWB (if the firmware update is enabled), for more information on this process see 8.

4.5 Operation of a tag

Initially a tag sleeps and periodically wakes up to listen for anchor's *Beacon* and *Almanacs* messages. It listens for a period before returning to sleep for before waking and trying again. The sleep period will initially be 10 s and will extend to 60 s.

When the tag receives valid *Beacons* and *Almanacs* messages, it firstly checks that it has compatible hardware version, firmware version and firmware size with the networked anchors. If the versions are incompatible then a firmware update process is initiated, as per 8. If the versions are compatible then the tag will continue with TWR slot reservation and TWR process.

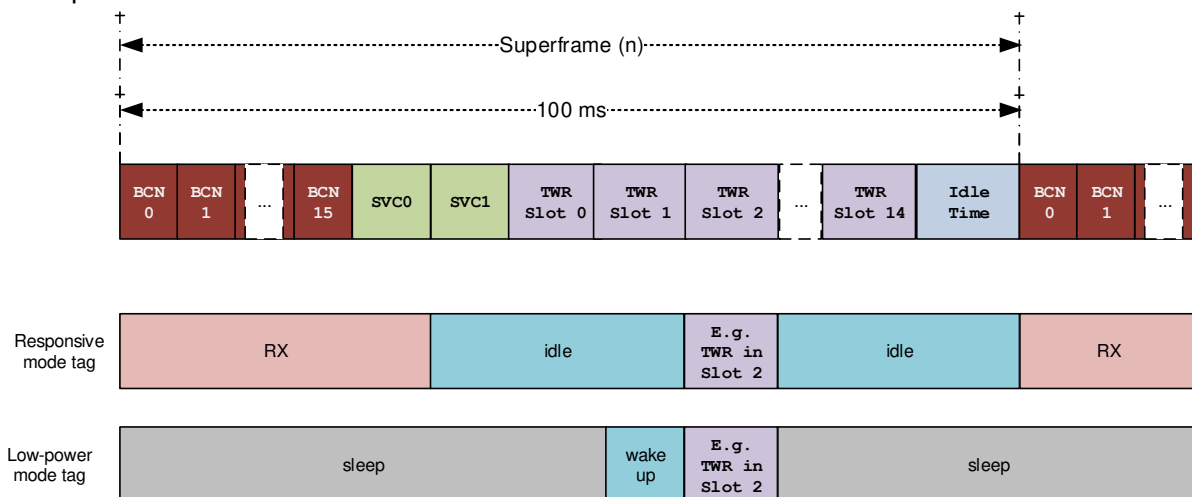


Figure 7: Tag operation in Low-power mode vs Responsive mode

A tag has two modes of operation:

- A *Responsive* mode – following the TWR exchange it will schedule the next listen period in which to listen for the *Beacons* of the superframe in which it has reserved the TWR exchange slot. The DW1000 will not be put into its low power state, but will remain in idle state (clock running). The nRF52832 MCU will be in sleep mode if no other tasks are running on the module. Typically used if Bluetooth is required.
- A *Low Power* mode – following the TWR exchange the DW1000 will be put into DEEPSLEEP and will be woken up prior to the next TWR exchange. The MCU will also be put into sleep with other components of the module except the RTC and

accelerometer (if accelerometer is enabled). The module is in the lowest power consumption mode. It will not listen to Beacons unless it moves out of the area in which the anchors it is currently ranging with are located. Once it leaves the area, the tag will proceed with TWR slot reservation as described below in 4.6.

4.6 TWR protocol and TWR slot reservation

The tag collects the ranging and data slot maps (which show the slot utilisation) from the *Beacon* messages of all anchors in range, and combines them to select a free ranging slot in the superframe in which to range. If there are no free ranging slots in the superframe it will try every 60 s to obtain new data and reserve a TWR slot. Each 100 ms superframe contains 15 ranging slots, each of which is dimensioned to allow the tag sufficient time to perform two-way ranging with 4 anchors, giving a maximum location rate capacity of 150 Hz. If all of the ranging slots are fully occupied, the system capacity is full and new tags will not be able to start ranging until the existing tags move out of the area or give up their slots.

The tag will use the reception time of the received *Beacons* to estimate its clock drift in comparison to the network time. A tag in Low Power mode will use time of reception of Poll message from the anchor to adjust its clock synchronisation. When it has the clock drift estimated the tag can start looking for a free TWR slot. When a free data slot is available it will initiate a location attempt by sending a *Group Poll* message.

The tag sends a *Group Poll* message containing: its location (ranging) period and a list of 4 addresses of the anchors it wishes to range to (and a bitmap with flags to indicate the anchors' seat numbers). The *Group Poll* message is a broadcast message so all anchors in range should receive it.

Each anchor listed in the group poll will (assuming it received the group poll) respond by sending a poll message in turn, with the transmit time being determined by the position (index, 0 to 3) of its address in the list. The tag will then send a *Response* message. This is followed by a *Final* message sent by each of the anchors in turn. Once the tag receives a final message from an anchor it can calculate the range to it. If the tag gets 3 or more valid ranges it will use its internal *Location Engine* to work out its location (relative to the anchors' coordinates). Figure 8 shows the individual TWR message slots inside the superframe structure.

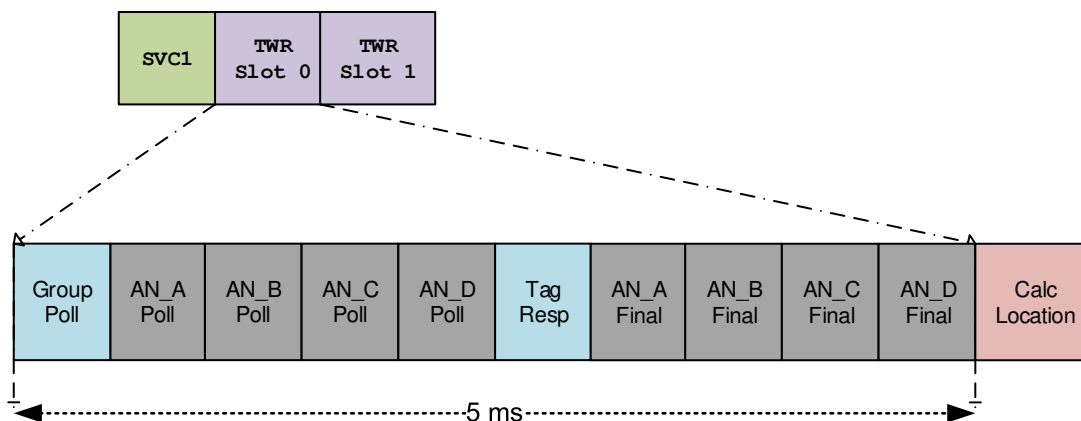


Figure 8: Superframe showing TWR frames

The tag sends its update rate in the *Group Poll* message, each of the 4 addressed anchors will send in their *Poll* messages their future free slots corresponding to the update rate of the tag. Using the information from the anchors (e.g. all the anchors report the future slot as

available) the tag requests the next TWR slot in the *Response* message. If no conflict is detected, the anchors will send confirmation of the data slot in the *Final* message. They will mark the slot as reserved so it will not be available for other tags.

The anchor stores slot occupancy in a TDMA bitmap. The length of the TDMA map is 60 s i.e. the maximum period which tag can reserve in advance (minimum location update rate).

4.7 TWR collision detection and resolution

DRTLS employs TDMA, which means that tags perform their two-way ranging to anchors in reserved slots, thus there should be no interference. However, as a tag moves (roaming) from one area to another or during initial picking of “free” slots, two tags might transmit at the same time. To help mitigate this, tags and anchors have a collision detection and resolution algorithm. If the number of the TWR measurements is lower than expected the tag could be experiencing following scenarios:

- The selected anchors might be out of range
- There could have been a collision with another tag
- There could have been collision detection at the anchor side

An anchor sends in one poll slot and receives in three other poll slots so it may see a message from an anchor that is not in the list of group poll and this means collision. If during the *Poll* phase the anchor received a *Poll* frame with different source and/or destination address than expected, or if it received a different type of message than expected, it will signal a collision. It will report this in the *Poll* or *Final* message.

The tag will signal a collision if a frame is received with different source and/or destination address than expected, or an unexpected message is received, or no *Poll* received or less than 3 *Polls/Finals* received. It will use this information to decide if it should keep using the ranging slot of it should give it up and try to look for a new data slot. The decision is based on following criteria:

- At the end of the *Poll* phase if the tag received less *Polls* than expected and collision was detected, then it will give up the slot if the amount of received *Polls* was less than 3 (minimum amount of distances to calculate a position) or the amount of next slot reservation confirmation is less than 3.
- If all the anchors confirmed the next slot then it will continue using it.
- If some anchors did not confirm the next slot then a collision is assumed and it will give up the slot.
- If no *Final* has been received due to a collision or a lost message, the tag will look for a new slot.

4.8 Tag's TWR Strategy

A tag collects information about anchors by listening to the *Beacon* messages. It will create a list of anchors from which it already received the positions. Then it will calculate distances to each of the anchors on the list, based on its current position (if it does not know its position it will use 0,0,0) it can then decide which anchors to choose for the next measurement using the following criteria:

- If possible choose an anchor from each quadrant, i.e. the tag will be surrounded by the anchors with whom it will range with. The tag is inside the polygon created by the selected anchors.
- Select the anchors which are nearest to the tag. It will keep using the selected anchors until it leaves the polygon or measurement with the selected anchors is no longer possible (TWR failed or collision detected).

Consider Figure 9, here T0 selects A1, A2, A9 and A7 as 4 anchors to range with. Each anchor is in a separate quadrant (Q1, Q2, Q3 and Q4).

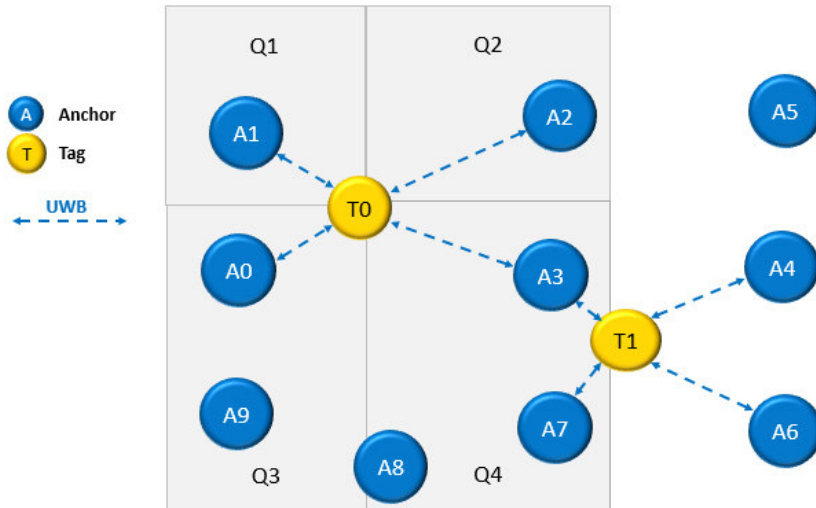


Figure 9: Tags choice of anchors for TWR

5 LOCATION ENGINE

The internal TWR location engine is used in tag mode to calculate an estimate of the tag's position using the two-way ranging results and known positions of the anchors selected for the ranging. A location estimate can be calculated with either 3 or 4 range results, i.e. can tolerate missing a response from any one of the four anchors selected for ranging and still calculate a new estimate of the tags location.

The location engine uses maximum likelihood estimation. When it has four ranging results the location engine creates sets of data (4 sets of 3 ranges). The sets are then being used to calculate possible tag positions. The implementation uses cache to speed up the estimation of the positions. If the anchors which are used for calculation has not been changed, the cached value will be used and there is no need to recalculate the initial matrices. The location engine then uses different criteria to choose or to combine them to calculate the estimation of position. The estimated position is verified by using measured distances. The positions which result in shorter distances than the measured are considered less accurate (multipath will result only in longer distances).

The location engine calculates the errors between the estimated positions and the real distance and removes the positions which have high errors. The final position is calculated using the selected estimated positions. The location engine will also report a quality factor (0-100) based on all of these criteria and the calculated errors.

A fixed moving average of the last 3 location results is used for the estimation of the final position.

6 POWER MANAGEMENT STRATEGY

The DWM1001 module FW when operating in default low-power tag mode uses power management strategy to help to keep the system and its components in lowest power mode between the ranging exchanges. The anchors are power efficient but not as power efficient as the tags, as they are continuously on and waiting to be involved in ranging with tags. It is assumed the anchors will be powered via mains power supply.

The DWM1001 onboard operating system (eCos) will register all the tasks with the Power Management function. When all of the OS tasks have completed their operation and are in idle mode, the Power Management will put the MCU into sleep mode without the RTOS tick. While any tasks are running, the RTOS tick is active and the MCU cannot be put into the sleep mode.

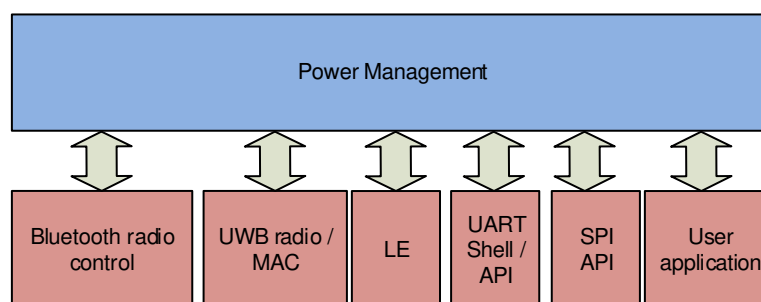


Figure 10: Main DWM1001 system components

6.1 Power Management control of main system components

The Power Management and main system components are shown in Figure 10. The Power Management controls:

- **UWB radio:** The use of DW1000 radio through its driver and the MAC state machine
- **LE:** Location engine operation, it is run as soon as the new range measurements are available, and then goes to idle.
- **Bluetooth radio:** The Bluetooth advertisements are transmitted on DWM1001 power up and when the user presses the Bluetooth wake up button. The advertisements are transmitted for 20 s. When another Bluetooth device (tablet) is connect to the module, the connection is maintained and the module (i.e. tag) will not be able to go to sleep. As soon as the device disconnects, the Bluetooth module will send advertisements for 20 s and if no connection is established, the Bluetooth module will be disabled and its resources released allowing the device to enter sleep state.
- **UART Shell:** Once *shell* is enabled the MCU will not go into sleep mode until the user runs the “quit” command, and disables the *shell*. That means the *shell* and its resources are released from the Power Management.
- **UART API:** Once UART API is enabled (a communication was initialised via UART from a host device – see [5]), The MCU will not go into sleep mode and the host needs to control the MCU sleep via an API call.
- **SPI API:** Once SPI API is enabled (a communication was initialised via UART from a host device – see [5]), The MCU will not go into sleep mode and the host needs to control the MCU sleep via an API call.
- **User Application:** A user can add own application into the DWM1001 FW. This is described in detail in [5]. This application has two options to register with the Power Management and influence when the MCU is sleeping/going into low power mode:

- By registering to receive location data - on reception of location data via the callback, a Power Management automatically waits for User Application to notify it via the API call (when finished) and then the Power Management can put the MCU into sleep mode if no other tasks are pending.
- By registering with the Power Management task and notifying it using API calls

6.2 *Wake-up sources*

To wake up the tag from low power (sleep) mode a number of signals are used:

- **RTC:** The RTC is used to wake-up the UWB radio and its MAC so that the device can be ready for the next ranging exchange.
- **Accelerometer:** If accelerometer detects movement it will wake up the device and the tag will change to using the non-stationary location rate.
- **UART RX GPIO:** The host can wake up the device with UART RX GPIO so that it can communicate with it via the UART APIs.
- **SPI CS GPIO:** The host can also use SPI CS signal to wake up the device.
- **User button:** The DWM1001 GPIO2 (e.g. if connected to a button e.g. DWM1001 – DEV) can also be used to wake up the device.

6.3 *Two location update rates*

To help reduce power consumption the tag has an option of two location update rates, the nominal and stationary. The onboard accelerometer is used to detect when the device is stationary and then the stationary rate will be used. If the accelerometer is disabled then only the nominal location rate will be used.

7 SCALABILITY

7.1 System Expansion - Anchors

The network infrastructure consists of anchor-initiators and anchors. The entire network uses a TDMA scheme, and thus all anchor nodes need to keep synchronised with the superframe timing of the initiator so they can send in their designated slots.

The network uses collision avoidance, collision detection and collision resolution techniques. Each anchor obtains a BCN slot/seat (4.2.2) and then can participate in the network and transmit *Beacon* and *Almanac* messages.

The system can be scaled to large network sizes but there are a number of scaling rules that must be followed to allow this.

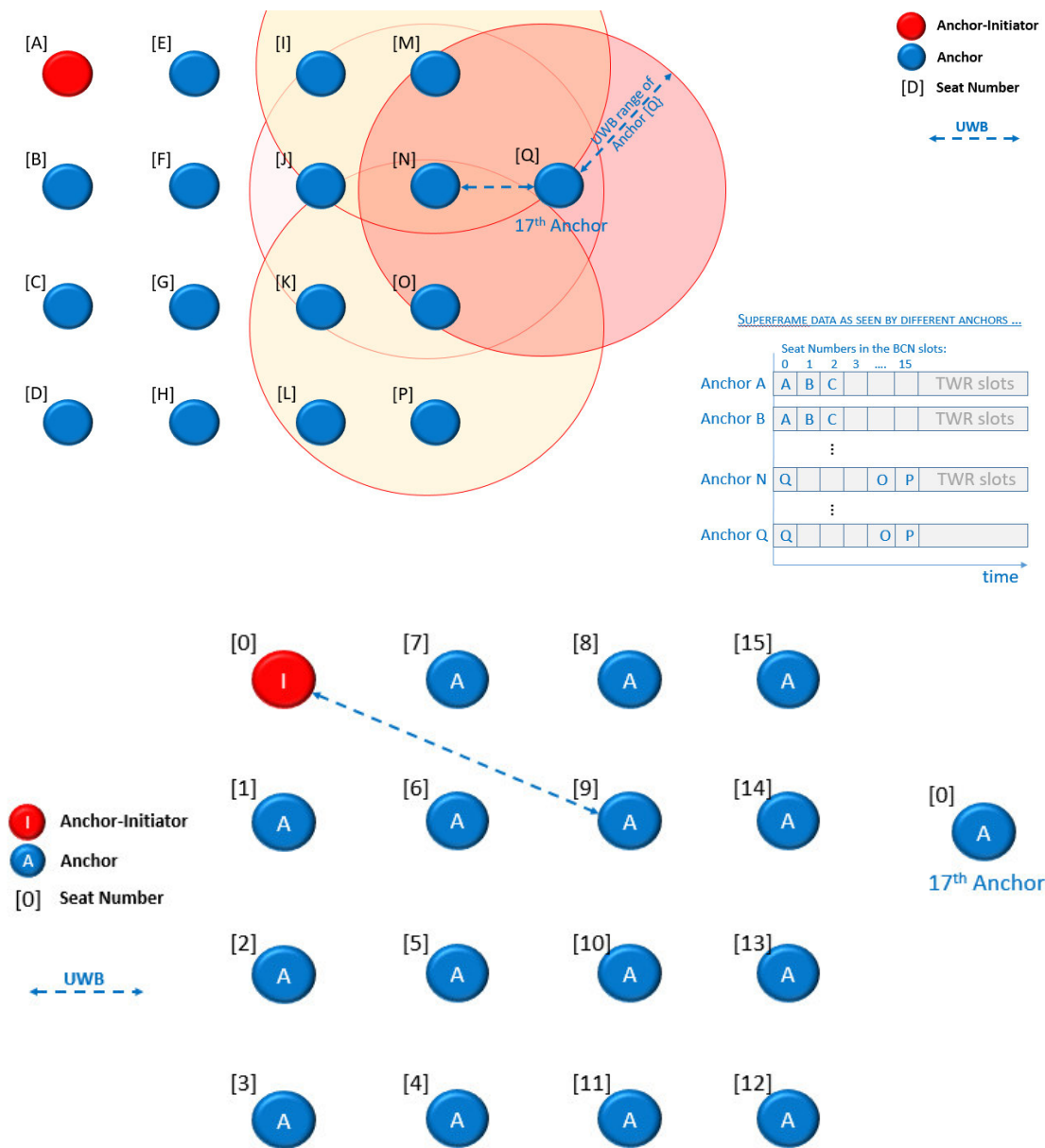


Figure 11: Scaling the DRTLS showing anchor's seat numbers

7.1.1 Scaling Rules

- Each anchor needs to be assigned a seat number between 0 and 15
- No anchor is allowed to hear 2 anchors with the same seat number
- All nodes must be synchronised with the superframe timing of the initiator

7.2 Installation Limitations

7.2.1 Limitation 1: Maximum number of anchor seats is 16

- If the system has 16 anchors or less there are no restrictions on anchor positioning (they can all be in range of each other)
- If a 17th (or more) anchor is required then it needs to re-use a seat number. This can only be achieved if the other anchors are spaced sufficiently far apart, such that no anchor hears two anchors with the same seat.
- Before allowing the 17th anchor to join, all other anchors that are within range of the 17th must confirm to it that a seat number is available (as per 4.2.1)
- For example, in the diagram above, if the anchor 9 is within range of both anchor 0 and the 17th anchor, then the 17th anchor may not use seat number 0 (since that would mean that anchor 9 would be able to see two anchors with the same seat number, i.e. 0)
- The implication of this restriction is that it is best to place anchors reasonably far apart to allow re-use of seat numbers i.e. do not over-populate a single space with too many anchors

7.2.2 Limitation 2: Maximum number of clock level is 127

- Each connected node will have its clock derived from the network initiator clock or from its neighboring anchor which is closer to the initiator.
- All nodes within range of the initiator will have a clock level of 1
- Nodes that are in range of nodes with clock level 1, but not in range of the initiator, will have a clock level of 2 and so on
- The highest allowed value of clock level is 127. This means that if a new anchor is trying to join, and it can hear *Beacons* from anchors that are already at clock level 127. It will not be able to join the network.

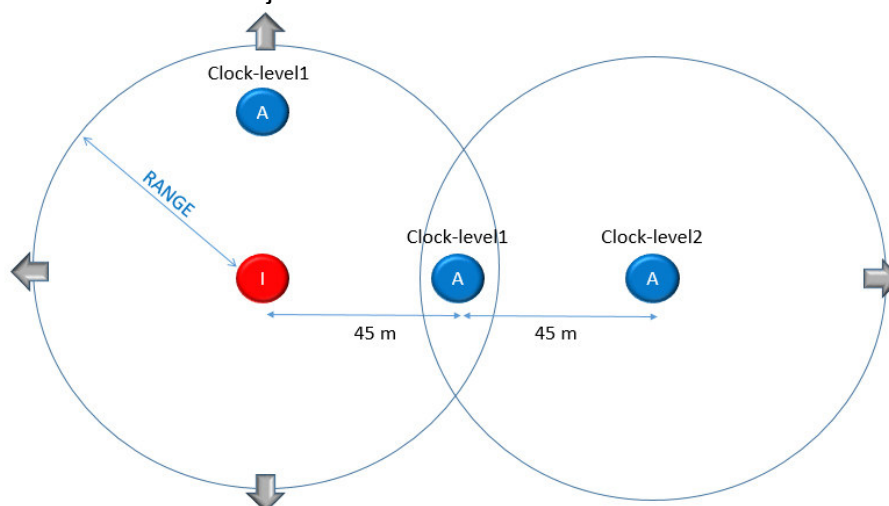


Figure 12: Scaling the DRTLs showing clock levels

7.3 System Expansion – Tags

The system is designed to have a 150 Hz system capacity e.g.

- 15 tags @ 10 Hz (max. location rate)
- 150 tags @ 1 Hz
- 300 tags @ 0.5 Hz
- 9000 tags @ 0.01667 (min. location rate)

Tags are assigned a specific TWR slot and range to up to 4 anchors. If all the slots (there are 15 slots in 1 superframe) have already been allocated to tags, then a new tag will be unable to obtain a slot in which to range. In a similar way that anchors spread over space can reuse seats when out of range of all other users of that seat, over a wider area ranging slots will similarly be able to be reused.

When setting tag update rate the system installer should consider what level of service will result if it is possible for tags to congregate in one area and should configure the maximum location rate to support that. Otherwise as tags congregate some may become unlocatable as they fail to get access to a ranging slot.

7.4 Network Coverage and Expansion

The system supports two network topologies: star and line. The figures below (Figure 13 and Figure 14) show the approximate area that can be covered if 40m LOS range is assumed between the anchors. Note in the network below all the nodes have to be within range of at least one routing anchor so that they have an uplink/downlink data connection to the gateway.

To cover larger areas further gateways (bridge nodes) and routing anchors need to be added, and system expanded accordingly.

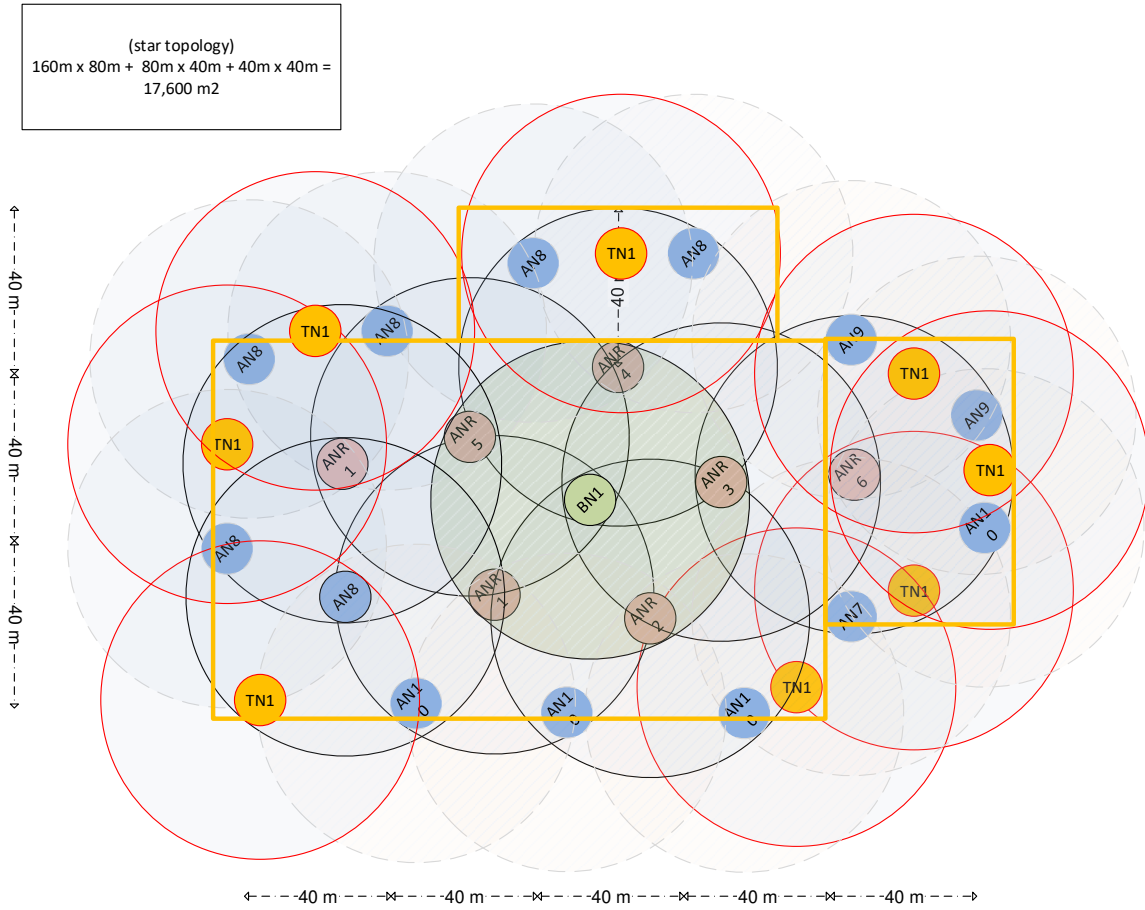


Figure 13: Single gateway area coverage with star topology

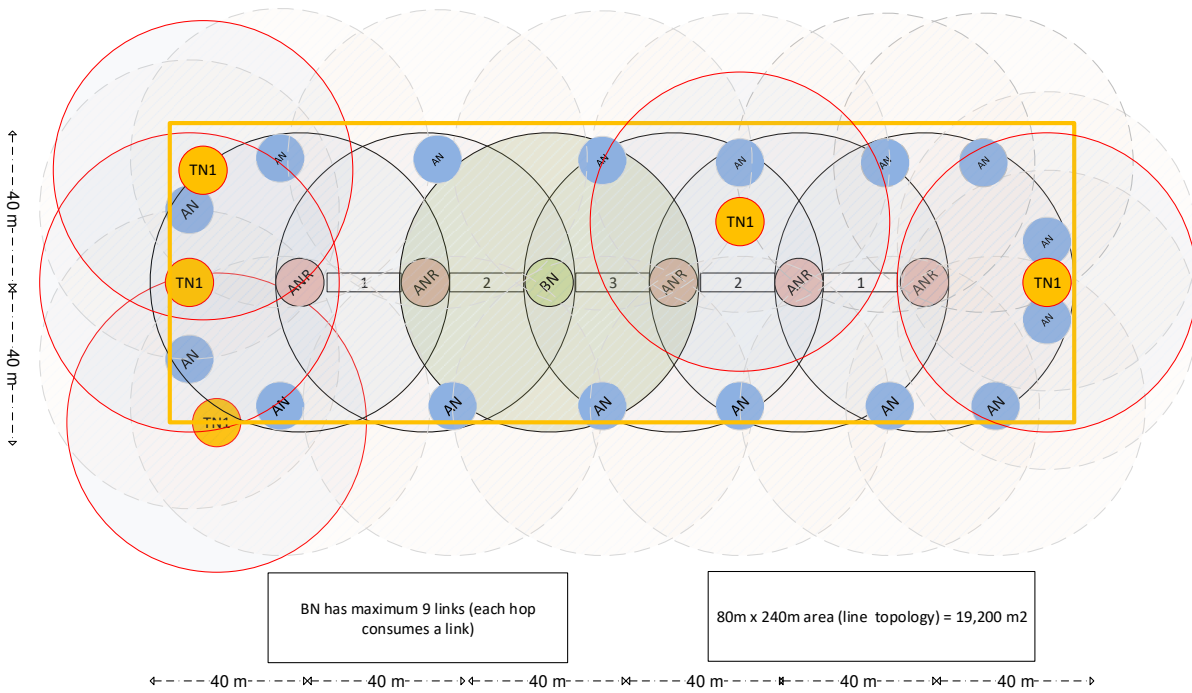


Figure 14: Single gateway area coverage with line topology

8 MEMORY USAGE AND FIRMWARE UPDATE

The DW1001 comes preloaded with DRTL5 firmware, which enables the building of a fully functioning RTLS without any further firmware change, however if the need arises it is possible to reprogram the module firmware in its flash memory. This process, called firmware update, is described in detail in DWM1001 Firmware User Guide [5].

The flash memory structure is shown in Figure 15. The area labelled FW2 contains the main functional block and application and area FW1 is just used for the update process.

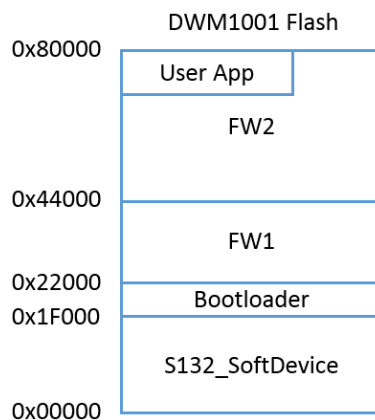


Figure 15: DWM1001 flash address map

8.1 Firmware Update

When the RTLS network is forming, the initiator anchor specifies the firmware version necessary for the network. When automatic FW update is enabled, any devices wishing to participate (join) the network must have the same firmware (version number and the checksum). If a new device does not have the correct firmware it will be updated as per the sub-sections below.

8.2 Firmware update initiated over Bluetooth

If one wants to update the entire network to a new firmware image while the network is operational, it is sufficient to just update the initiator via Bluetooth. The initiator will then propagate the new firmware to all of the other devices over the UWB radio link automatically.

Note, as the initiator is updated first, it will restart the network and as each device re-joins the network its firmware will be updated. Thus, during the FW update the nodes which are performing the update will be “offline”.

8.3 Firmware update via UWB

When automatic FW update is enabled, every device wanting to join the network needs to have the same firmware version. Each new joining device listens to the *Almanacs* to obtain the system hardware and firmware information. If a firmware update is needed the new device will choose a nearby anchor with free TWR/data slot and will send a *Firmware Update Data Request* message (9.1.6).

The anchor which received the request to supply the firmware, checks the request and if no firmware update process is running will start sending the firmware update data to the

requesting anchor over UWB. The firmware data messages are broadcast so any node in firmware update state can receive and process the data.

The receiving node stores the new firmware image data in an intermediate buffer and when the buffer contains the size of internal flash page size, it will be written to the internal flash. Note, during the firmware update, Bluetooth SoftDevice is disabled because writing to internal flash via SoftDevice is very slow (up to 500 ms per page). When the SoftDevice is disabled, a page can be written within ~120 ms.

If the receiving device misses some data (due to failed frame reception), it will request the same data again starting the offset where it stopped in the previous attempt. The cycle will repeat until the requesting node would receive all firmware data. Then it will run a checksum control of the whole firmware and if the checksum values is correct, it will enable the non-volatile register to boot from the other firmware (FW1/FW2) reset. If the checksum failed, it will repeat the whole process again.

Firstly the unit will be running from FW2 and will update FW1, then it will reset and update FW2.

8.4 Manual firmware update

When automatic FW update is disabled, then the user can individually update firmware in a device via Decawave DRTLS Manager or via the SWD connection.

9 APPENDIX: FRAME FORMATS

9.1 IEEE 802.15.4 frame

The system uses standard frames as defined by IEEE 802.15.4 on the MAC layer. The standard defines multiple frame types, two out of which are used in the system: data frame and acknowledgement (ACK) frame. Table 2 shows the data frame structure, and Table 3 ACK frame.

Table 2: IEEE 802.15.4 Data Frame Structure

Field	Octets	Description
Frame control	2	As defined in the IEEE 802.15.4
Sequence number	1	Modulo 256 sequence number
PAN ID	2	PAN ID
Destination address	2	Address of destination node, or 0xFFFF if this is a broadcast frame (e.g. <i>Group Poll</i> message)
Source address	2	Node's own address is fixed and is the lower 16-bits of the generated 64-bit address (derived as 0xDECA + 28 bits of MCU Unique ID + 20 bit of DW1000 Part ID)
Payload	1-1014	Note: the first byte of the payload always denotes the frame type which is followed by the message content as specified in tables below. Message ID codes: 0x10 - UWBMAC_FRM_TYPE_BCN 0x11 - UWBMAC_FRM_TYPE_SVC 0x12 - UWBMAC_FRM_TYPE_CL_JOIN 0x13 - UWBMAC_FRM_TYPE_CL_JOIN_CFM 0x18 - UWBMAC_FRM_TYPE_POS 0x21 - UWBMAC_FRM_TYPE_FWUP_DATA_REQ 0x22 - UWBMAC_FRM_TYPE_FWUP_DATA 0x23 - UWBMAC_FRM_TYPE_ALMA 0x30 - UWBMAC_FRM_TYPE_TWR_GRP_POLL 0x31 - UWBMAC_FRM_TYPE_TWR_POLL 0x32 - UWBMAC_FRM_TYPE_TWR_RESP 0x33 - UWBMAC_FRM_TYPE_TWR_FINAL
Frame check sequence	2	DW1000 automatically generated FCS

The payload of the data frame (see Figure 16) can contain one or more of the messages specified in the sub-section below. Generally, only one message is sent per data frame, the only exception being the Beacon message, which can be followed by an extra message (e.g. Join Confirmation or Position messages).

Table 3: IEEE 802.15.4 ACK frame

Field	Octets	Description
Frame control	2	As defined in the IEEE 802.15.4
Sequence number	1	Modulo 256 sequence number copied from the data frame being acknowledged
Frame check sequence	2	DW1000 automatically generated FCS

The IEEE MAC header and payload is shown in Figure 16.

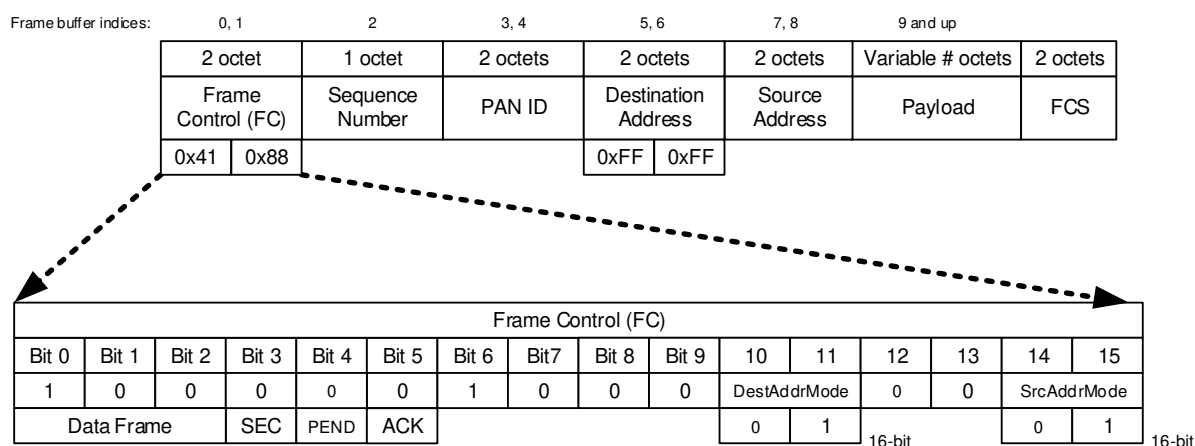


Figure 16: IEEE 802.15.4 MAC frame format, with e.g. broadcast address and

9.1.1 Beacon message

These messages are sent by the anchors in the BCN slots in the superframe. These messages have the IEEE 802.15.4 data frame format with payload specified in the following table. The *Beacon* messages contain information about the current superframe and network slot usage (i.e. which TWR slots are currently used by tags/anchors).

Payload Field	Octets	Description
Message ID	1	0x10 - UWBMAC_FRM_TYPE_BCN
Session ID	1	Random number which the Anchor Initiator generates during network initialisation. All joined anchors must have the same Session ID
Cluster flags	1	Bit field where following flags can be set: <ul style="list-style-type: none"> AUTOPOS - auto-positioning is in progress throughout the network, can be used to auto-locate the anchors in range of the initiator

		<ul style="list-style-type: none"> • DAT_DL - down-link data will be sent in this superframe • SVC_DL - service slot in this superframe is down-link • SVC_UL - service slot in this superframe is up-link • INIT - node is initiator • EXT - when set, there is another message following the beacon message, in the payload of the same IEEE 802.15.4 data frame; length of the extra frame is specified in the payload immediately following the beacon message
Cluster slot number	1	Cluster slot number ~ seat number
Cluster frame number	1	Cluster message number - used for superframe synchronisation
Cluster map	2	Bitmap indicating occupied seats visible by the sending anchor (a set bit indicates occupied seat, e.g. 0x0081 means seats 7 and 0 are occupied)
Neighbour cluster map	2	Bitmap indicating occupied seats visible by the sending anchor's neighbour (a set bit indicates occupied seat, e.g. 0x8002 means seat 15 and 1 are occupied)
Data slot map	2	Indicates which TWR data slots of the sending anchor are occupied

9.1.2 Join request message

These messages are sent by the anchors, during the Service interval (SVC slots) inside the superframe, as a request to join the network. These are the IEEE 802.15.4 data frames with message payload specified in the following table. These are sent to networked anchors as a request to join the network.

Payload Field	Octets	Description
Message ID	1	0x12 - UWBMAC_FRM_TYPE_CL_JOIN
Hardware version	4	0xDE00002A = DWM1001 hardware version.
Firmware version	4	0x01010001 = 01.01.00.0.1 = Major.Minor.Patch.Reserved.FirmwareVariant
Firmware checksum	4	Firmware checksum - CRC32
Options	4	Bitmap indicating node capabilities
Cluster seat	1	Requesting cluster seat

9.1.3 Join confirmation message

These messages are sent by the anchors after a join request is received. These messages are sent following the *Beacon* message in the same IEEE 802.15.4 data frame (see the EXT flag in the *Beacon* message) and have the format specified in the following table. They are sent in a response to join request to allocate a seat to the anchor trying to join the network.

Payload Field	Octets	Description
Message ID	1	0x13 - UWBMAC_FRM_TYPE_CL_JOIN_CFM
Address	2	Locked address of the joining node
Cluster lock	1	Lock counter (decrementing). 0 means the next <i>Beacon</i> will not contain join confirmation unless a new request is received
Cluster seat	1	Confirming allocated seat number or 0xff indicating no seat has been allocated

9.1.4 Almanac message

These are sent by the networked anchors, during the Service interval (SVC slots) inside the superframe. These are IEEE 802.15.4 data frames with payload specified in the following table. Each superframe can only carry a single *Almanac* message, so each of the 16 anchors of the cluster takes its turn to send its almanac once every 16 superframes according to its seat number. They provide the node FW versions and node's capabilities.

Payload Field	Octets	Description
Message ID	1	0x23 - UWBMAC_FRM_TYPE_ALMA
Flags	1	Special flags, e.g. firmware update force.
Hardware version	4	Hardware version of the sending node
Firmware version	4	Firmware version of the sending node
Firmware 1 size	4	Firmware 1 size of the sending node
Firmware 2 size	4	Firmware 2 size of the sending node
Firmware 1 checksum	4	Firmware 1 checksum of the sending node
Firmware 2 checksum	4	Firmware 2 checksum of the sending node
Node ID	8	Complete 64-bit address of the sending node
Node option	4	Bitmap indicating node capabilities

9.1.5 Service message

The Service messages are used for sending service commands and status between the networked devices. E.g. collision reports. These are sent in the SVC slots. These are IEEE 802.15.4 data frames with payload as specified in the following table.

Payload Field	Octets	Description
Message ID	1	0x11 - UWBMAC_FRM_TYPE_SVC
Code	1	Service code
Argc	1	Number of argument octets
Argv	0-32	Arguments

9.1.6 Firmware Update Data Request message

These messages are sent by devices which need to update their FW. The request is sent to one of the networked anchors. These are IEEE 802.15.4 data frames with payload as specified in the following table.

Payload Field	Octets	Description
Message ID	1	0x21 - UWBMAC_FRM_TYPE_FWUP_DATA_REQ
Offset	4	Requesting data offset
Size	4	Requesting data size
Hardware version	4	Requesting hardware version
Firmware version	4	Requesting firmware version
Firmware size	4	Requesting firmware size
Firmware checksum	4	Requesting firmware checksum

9.1.7 Firmware Update Data message

These messages contain the Firmware image data and are sent from the networked anchor to the device that has requested the update. These are IEEE 802.15.4 data frames with payload as specified in the following table.

Payload Field	Octets	Description
Message ID	1	0x22 - UWBMAC_FRM_TYPE_FWUP_DATA
Offset	4	Firmware offset of the sending data
Length	4	Length of the sending data

Buffer	1-512	Content of the sending firmware buffer
--------	-------	--

9.1.8 Position message

The *Position* message is sent as a part of the extended *Beacon* message, it contains the anchor coordinates. The message payload is as specified in the following table.

Payload Field	Octets	Description
Message ID	1	0x18 - UWBMAC_FRM_TYPE_POS
X	4	X coordinate (of the anchor position)
Y	4	Y coordinate (of the anchor position)
Z	4	Z coordinate (of the anchor position)

9.1.9 Group Poll message

This message is sent from a device that wants to initiate a TWR exchange. This will typically be a tag but it can also be an anchor (e.g. during auto-positioning process). These are IEEE 802.15.4 data frames with payload as specified in the following table. This is sent as a broadcast so all devices can receive it.

Payload Field	Octets	Description
Message ID	1	0x30 - UWBMAC_FRM_TYPE_TWR_GRP_POLL
Flags	2	Bitmap indicating the anchors to range with. Anchor uses this bit map to figure out if its seat corresponds with the indicated bit. If it does, it will continue to look for its address in the Address field below else the frame will be ignored.
Update period	2	Requesting update period in mili-seconds
Address	8	Four 16-bit addresses of the anchors to range with
Sequence number	1	TWR sequence number
Quality factor	1	Position quality factor
X	4	X coordinate in meters (of the last calculated position)
Y	4	Y coordinate in meters (of the last calculated position)
Z	4	Z coordinate in meters (of the last calculated position)

9.1.10 Poll message

An anchor receiving a *Group Poll* nominating it as one of the four addressed anchors will respond with a *Poll* message. This is a part of the double-sided TWR protocol. These are IEEE 802.15.4 data frames with payload as specified in the following table. These are addressed to the particular device which sent the *Group Poll*.

Payload Field	Octets	Description
Message ID	1	0x31 - UWBMAC_FRM_TYPE_TWR_POLL
Flags	1	TWR_FL_ECOLL_INR - Collision detected at TWR Initiator TWR_FL_ECOLL_RSP - Collision detected at TWR Responder TWR_FL_ETYPE_INR - Incorrect frame type at TWR Initiator TWR_FL_ETYPE_RSP - Incorrect frame type at TWR Responder
Data slot map	2	Map of TWR slots available in the future period

9.1.11 Response message

A tag which receives at least one *Poll* will respond to the with a *Response* message, as per the TWR protocol. These are IEEE 802.15.4 data frames with payload as specified in the following table. The *Response* messages are also sent as broadcasts. As they are a reply to the group of anchors the tag is ranging with.

Payload Field	Octets	Description
Message ID	1	0x32 - UWBMAC_FRM_TYPE_TWR_RESP
Flags	1	TWR_FL_SL_REQ - Indicate future TWR slot request for the next ranging interaction
Data slot	1	Requesting TWR slot

9.1.12 Final message

An anchor which receives the *Response* message will respond to the tag with a *Final* message, as per the TWR protocol. These are IEEE 802.15.4 data frames with payload as specified in the following table. These will be addressed to the particular device that sent the *Response* message.

Payload Field	Octets	Description
Message ID	1	0x33 - UWBMAC_FRM_TYPE_TWR_FINAL
Flags	1	TWR_FL_SL_CFM - indicates slot confirmation for the next update
TX poll timestamp	5	Transmit time of the Poll message in DW system time units
RX response timestamp	5	Receive time of the Response message in DW system time units
TX final timestamp	5	Transmit time of the Final message in DW system time units

10 REFERENCES

10.1 Listing

Reference is made to the following documents in the course of this document:

Table 4: Table of References

Ref	Author	Version	Title
[1]	Decawave	Current	DWM1001 Product Brief
[2]	Decawave	Current	DWM1001-DEV Product Brief
[3]	Decawave	Current	MDEK1001 Product Brief
[4]	Decawave	Current	MDEK1001 User Manual
[5]	Decawave	Current	DWM1001 Firmware User Guide

11 DOCUMENT HISTORY

11.1 Revision History

Table 5: Document History

Revision	Date	Description	Revised By
1.0	18 th December 2017	Release for publication	DB, ZS

11.2 Major changes

Revision 1.00

Page	Change Description
All	Initial Release

12 ABOUT DECAWAVE

Decawave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4-2011 UWB standard. This device is the first in a family of parts that will operate at data rates of 110 kbps, 850 kbps and 6.8 Mbps.

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

Further Information

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

Decawave Ltd
Adelaide Chambers
Peter Street
Dublin
D08 T6YA
Ireland
t: +353 1 6975030
e: sales@decawave.com
w: www.decawave.com

DWM1001 FIRMWARE APPLICATION PROGRAMMING INTERFACE (API) GUIDE

**USING API FUNCTIONS TO
CONFIGURE AND PROGRAMME
DWM1001 MODULE**

This document is subject to change without notice

TABLE OF CONTENTS

DISCLAIMER	7
1 INTRODUCTION AND OVERVIEW	10
1.1 DWM1001 MODULE AND THE FIRMWARE.....	10
1.2 API AND ITS GUIDE.....	10
2 GENERAL API DESCRIPTIONS	11
2.1 EXTERNAL INTERFACE USAGE.....	11
2.2 LOW POWER MODE WAKE-UP MECHANISM.....	11
2.3 TLV FORMAT.....	11
2.4 DWM1001 THREADS.....	11
2.5 API VIA BLE INTERFACE.....	12
2.6 API VIA SPI INTERFACE.....	12
2.6.1 <i>DWM1001 SPI overview</i>	12
2.6.2 <i>SPI Scheme: normal TLV communication</i>	15
2.6.3 <i>SPI Example: normal TLV communication - dwm_gpio_cfg_output</i>	16
2.6.4 <i>SPI Scheme: TLV communication using data ready pin</i>	17
2.6.5 <i>SPI error recovery mechanism</i>	18
2.7 API VIA UART INTERFACE.....	19
2.7.1 <i>DWM1001 UART overview</i>	19
2.7.2 <i>UART TLV Mode</i>	19
2.7.3 <i>UART scheme: TLV mode communication</i>	20
2.7.4 <i>UART example: TLV mode communication</i>	21
2.7.5 <i>UART scheme: Shell mode communication</i>	21
2.7.6 <i>UART example: Shell Mode communication</i>	22
2.8 GPIO SCHEME: DWM1001 NOTIFIES FOR STATUS CHANGE.....	22
2.9 API FOR ON-BOARD C CODE DEVELOPERS.....	23
3 GENERIC API INFORMATION	24
3.1 USED TERMINOLOGY.....	24
3.2 LITTLE ENDIAN.....	24
3.3 FIRMWARE UPDATE.....	24
3.4 FREQUENTLY USED TLV VALUES.....	25
3.4.1 <i>err_code</i>	25
3.4.2 <i>position</i>	25
3.4.3 <i>gpio_idx</i>	25
3.4.4 <i>gpio_value</i>	25
3.4.5 <i>gpio_pull</i>	26
3.4.6 <i>fw_version</i>	26
3.4.7 <i>cfg_tag</i>	26
3.4.8 <i>cfg_anchor</i>	26
3.4.9 <i>cfg_node</i>	27
4 API FUNCTION DESCRIPTIONS	28
4.1 LIST OF API FUNCTIONS.....	28
4.2 USAGE OF THE APIS.....	29
4.3 DETAILS OF THE API FUNCTIONS.....	29

4.3.1	<i>dwm_pos_set</i>	30
4.3.2	<i>dwm_pos_get</i>	31
4.3.3	<i>dwm_upd_rate_set</i>	32
4.3.4	<i>dwm_upd_rate_get</i>	33
4.3.5	<i>dwm_cfg_tag_set</i>	34
4.3.6	<i>dwm_cfg_anchor_set</i>	36
4.3.7	<i>dwm_cfg_get</i>	37
4.3.8	<i>dwm_sleep</i>	39
4.3.9	<i>dwm_loc_get</i>	40
4.3.10	<i>dwm_baddr_set</i>	42
4.3.11	<i>dwm_baddr_get</i>	43
4.3.12	<i>dwm_reset</i>	44
4.3.13	<i>dwm_ver_get</i>	45
4.3.14	<i>dwm_gpio_cfg_output</i>	46
4.3.15	<i>dwm_gpio_cfg_input</i>	47
4.3.16	<i>dwm_gpio_value_set</i>	48
4.3.17	<i>dwm_gpio_value_get</i>	49
4.3.18	<i>dwm_gpio_value_toggle</i>	50
4.3.19	<i>dwm_status_get</i>	51
4.3.20	<i>dwm_int_cfg</i>	52
4.3.21	<i>dwm_gpio_irq_cfg</i>	53
4.3.22	<i>dwm_gpio_irq_dis</i>	54
4.3.23	<i>dwm_i2c_read</i>	55
4.3.24	<i>dwm_i2c_write</i>	56
4.3.25	<i>dwm_evt_cb_register</i>	57
5	SHELL COMMANDS	59
5.1	USAGE OF UART SHELL MODE.....	59
5.2	?.....	59
5.3	HELP.....	60
5.4	QUIT.....	60
5.5	GC.....	60
5.6	GG.....	61
5.7	GS.....	61
5.8	GT.....	61
5.9	F.....	61
5.10	PS.....	61
5.11	PMS.....	62
5.12	RESET.....	62
5.13	UT.....	62
5.14	FRST.....	62
5.15	TWI.....	62
5.16	AID.....	63
5.17	AV.....	63
5.18	LES.....	63
5.19	LEC.....	63
5.20	LEP.....	63
5.21	SI.....	64
5.22	NMG.....	64
5.23	NMO.....	64

5.24	NMP	65
5.25	NMA	65
5.26	NMI	66
5.27	NMT	66
5.28	NMTL	67
5.29	BPC	67
5.30	LA	67
5.31	STG	68
5.32	STC	69
5.33	TLV	69
5.34	AURS	69
5.35	AURG	69
5.36	APG	70
5.37	APS	70
5.38	ACAS	70
5.39	ACTS	70
6	APPENDIX A – TLV TYPE LIST	71
7	APPENDIX B – BIBLIOGRAPHY	72
8	DOCUMENT HISTORY	73
9	ABOUT DECAWAVE	74

List of Tables

TABLE 1 TLV FORMAT DATA EXAMPLE	11
TABLE 2 API REQUEST FUNCTION LIST	28
TABLE 3 API EXAMPLES LOCATION	29
TABLE 4 DWM1001 TLV TYPE LIST	71
TABLE 5: DOCUMENT HISTORY.....	73

List of Figures

FIGURE 1 DWM1001 SPI WORK FLOW	13
FIGURE 2 SPI SCHEME: NORMAL TLV COMMUNICATION	15
FIGURE 3 SPI EXAMPLE: NORMAL TLV COMMUNICATION.....	16
FIGURE 4 SPI SCHEME: TLV COMMUNICATION USING DATA READY PIN	17
FIGURE 5 DWM1001 UART WORK FLOW	19
FIGURE 6 UART SCHEME: TLV MODE COMMUNICATION	20
FIGURE 7 UART EXAMPLE: TLV MODE COMMUNICATION	21
FIGURE 8 UART SCHEME: SHELL MODE COMMUNICATION	22
FIGURE 9 UART EXAMPLE: SHELL MODE COMMUNICATION.....	22
FIGURE 10 GPIO SCHEME: DWM1001 NOTIFIES HOST DEVICE OF STATUS CHANGE, USING GPIO.....	23

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 © 2017 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

- (1) This Disclaimer applies to the software provided by Decawave Ltd. (“Decawave”) in support of its DWM1001 module product (“Module”) all as set out at clause 3 herein (“Decawave Software”).
- (2) Decawave Software is provided in two ways as follows: -
 - (a) pre-loaded onto the Module at time of manufacture by Decawave (“Firmware”);
 - (b) supplied separately by Decawave (“Software Bundle”).
- (3) Decawave Software consists of the following components (a) to (d) inclusive:
 - (a) The **Decawave Positioning and Networking Stack** (“PANS”), available as a library accompanied by source code that allows a level of user customisation. The PANS software is pre-installed and runs on the Module as supplied, and enables mobile “tags”, fixed “anchors” and “gateways” that together deliver the DWM1001 Two-Way-Ranging Real Time Location System (“DRTLS”) Network.
 - (b) The **Decawave DRTLS Manager** which is an Android™ application for configuration of DRTLS nodes (nodes based on the Module) over Bluetooth™.
 - (c) The **Decawave DRTLS Gateway Application** which supplies a gateway function (on a Raspberry Pi ®) routing DRTLS location and sensor data traffic onto an IP based network (e.g. LAN), and consists of the following components:
 - DRTLS Gateway Linux Kernel Module
 - DRTLS Gateway Daemon
 - DRTLS Gateway MQTT Broker
 - DRTLS Gateway Web Manager
 - (d) **Example Host API functions**, also designed to run on a Raspberry Pi, which show how to drive the Module from an external host microprocessor.
- (4) The following third party components are used by Decawave Software and are incorporated in the Firmware or included in the Software Bundle as the case may be: -
 - (a) The PANS software incorporates the Nordic SoftDevice S132-SD-v3 version 3.0.0 (production) which is included in the Firmware and is also included in the Software Bundle;
 - (b) The PANS software uses the eCos RTOS which is included in the Software Bundle. The eCos RTOS is provided under the terms of an open source licence which may be found at: <http://ecos.sourceware.org/license-overview.html>;
 - (c) The PANS software uses an open source CRC-32 function from FreeBSD which is included in the Software Bundle. This CRC-32 function is provided under the terms of the BSD licence which may be found at: <https://github.com/freebsd/freebsd/blob/386ddae58459341ec567604707805814a2128a57/COPYRIGHT>;

- (d) The Decawave DRTLS Manager application uses open source software which is provided as source code in the Software Bundle. This open source software is provided under the terms of the Apache Licence v2.0 which may be found at <http://www.apache.org/licenses/LICENSE-2.0>;
- (e) The Decawave DRTLS Gateway Application uses the following third party components: -
 - (i) The Linux Kernel which is provided as source code in the Software Bundle. The Linux Kernel is provided under the terms of the GPLv2 licence which may be found at: <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html> and as such the DWM1001 driver component of the DRTLS Gateway Application is provided under the same license terms;
 - (ii) The three.js JavaScript library, the downloadable version of which is available here <https://threejs.org/>, is provided under the terms of the MIT Licence which may be found at <https://opensource.org/licenses/MIT>.

Items (a), (b), (c), (d) and (e) in this section 4 are collectively referred to as the “Third Party Software”

- (5) Decawave Software incorporates source code licensed to Decawave by Leaps s.r.o., a supplier to Decawave, which is included in the Firmware and the Software Bundle in binary and/or source code forms as the case may be, under the terms of a license agreement entered into between Decawave and Leaps s.r.o.
- (6) Decawave hereby grants you a free, non-exclusive, non-transferable, worldwide license without the right to sub-license to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating Decawave Software, to modify Decawave Software or incorporate Decawave Software in other software and to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating such modified or incorporated software PROVIDED ALWAYS that the use by you of Third Party Software as supplied by Decawave is subject to the terms and conditions of the respective license agreements as set out at clause 4 herein AND PROVIDED ALWAYS that Decawave Software is used only in systems and products based on Decawave semiconductor products. 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.
- (7) Downloading, accepting delivery of or using Decawave Software indicates your agreement to the terms of (i) the license granted at clause 6 herein, (ii) the terms of this Disclaimer and (iii) the terms attaching to the Third Party Software. If you do not agree with all of these terms do not download, accept delivery of or use Decawave Software.
- (8) 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 AND DECAWAVE ACCEPTS NO LIABILITY WHATSOEVER FOR SUCH DECISION.

- (9) DECAWAVE SOFTWARE IS PROVIDED "AS IS". DECAWAVE MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO DECAWAVE SOFTWARE OR USE OF 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.
- (10) 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 DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE. 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.
- (11) 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.
- (12) Decawave reserves the right to make corrections, enhancements, improvements and other changes to its software, including Decawave Software, at any time.

Mailing address: Decawave Ltd.,
Adelaide Chambers,
Peter Street,
Dublin D08 T6YA
IRELAND.

Copyright (c) 15 November 2017 by Decawave Limited. All rights reserved. All trademarks are the property of their respective owners.

1 INTRODUCTION AND OVERVIEW

1.1 DWM1001 module and the firmware

The DWM1001 module is a radio transceiver module integrating the Nordic Semiconductor nRF52 MCU and Decawave's DW1000 IC. The nRF52 MCU, which has Bluetooth v4.2 protocol stack implemented [1], is acting as the main processor of the DWM1001 module. The DW1000 IC part, which has the UWB physical layer defined in IEEE 802.15.4-2011 standard [2], is acting as the UWB radio module controlled by the nRF52 MCU.

Decawave provides a pre-built firmware library, the "Positioning and Networking stack" (PANS) library, in the DWM1001 module which runs on the nRF52 MCU. The firmware provides the Application Programming Interface (API) for users to use their own host devices to communicate with the DWM1001 module, namely the PANS API. The PANS API essentially is a set of functions providing a means to communicate with the nRF52 MCU to drive the module through the PANS library on application level without having to deal with the details of accessing the DW1000 IC part and other peripherals directly through its SPI/I2C interface register set. The detailed information of the firmware is introduced in the DWM1001 Firmware User Guide [3].

1.2 API and its guide

The PANS APIs are a set of functions. Each individual API function may be accessed through various communication interfaces providing flexibility to developers in using the DWM1001 module and integrating it into their systems. The API accesses mainly come in as two types:

- 1) External access API: via UART, SPI and BLE.
- 2) Integrated access API: via on-board user app (C code).

The detailed introduction to the flow with examples of how the API can be used is introduced in [3].

This document, "[DWM1001 API Guide](#)", specifies the PANS API functions themselves, providing descriptions of each of the API functions in detail in terms of its parameters, functionality and utility. Users can use the PANS API to configure each individual DWM1001 module. To setup a location system with multiple DWM1001 modules, users should refer to the DWM1001 System Guide [4].

This document relates to: ["DWM1001 PANS Library Version 1.1.5"](#)

2 GENERAL API DESCRIPTIONS

2.1 External Interface usage

The external interfaces, including the UART, the SPI and the BLE, are used by the external APIs in the PANS library for Host connection. The on-board user application through C code API cannot make use of the external interfaces due to compatibility reasons.

2.2 Low power mode wake-up mechanism

As provided in the PANS library, the DWM1001 module can work in a low power mode. In the low power mode, the module puts the API related threads into “sleep” state when the API is not currently being communicated by host devices. In this case, the host device needs to wake up the module through external interfaces before the real communication can start.

For UART interface, the host device needs to send a single byte first to wake up the module.

For SPI interface, putting the chip select pin to low level wakes up the module, i.e. no extra transmission is needed.

After the API transmission has finished, the host device needs to put the module back to “sleep” state to save power, as introduced in section 4.3.8 and section 5.4.

2.3 TLV format

TLV format, the Type-Length-Value encoding, is used in the DWM1001 module API. Data in TLV format always begins with the type byte, followed by the length byte, and then followed by a variable number of value bytes [0 to 253] as specified by the length byte. Table 1 shows an example of TLV format data, in which the type byte is 0x28, the length byte is 0x02, and as declared by the length byte, the value field is of two bytes: 0x0D and 0x01.

In DWM1001 firmware, both SPI and UART APIs use TLV format for data transmission. Users should refer to the type list for the meaning of type bytes (see Section 6). And for each specific command or response, the value field is of different length to provide the corresponding information.

Table 1 TLV format data example

TLV request			
Type	Length	Value	
		gpio_idx	gpio_value
0x28	0x02	0x0d	0x01

2.4 DWM1001 threads

In the DWM1001 firmware system, there are many threads, including SPI, BLE, UART, Generic API, User App and other threads. Each thread deals with specific tasks:

The SPI, BLE and UART threads control the data transmission with external devices. They don't parse the requests they've received. All received requests are sent to the Generic API thread.

The Generic API thread is a parser of the received requests. It judges whether the incoming request is valid. If valid, the firmware goes on to prepare the corresponding data as response; if invalid, the firmware uses error message as response. Then the Generic API thread runs the `call_back()` function which sends the prepared response message back to the thread where the request comes from.

The on-board user application thread is an independent thread for the users to add their own functionalities. The entrance is provided in the `dwm\example\` folder in the DWM1001 on-board package. An example project is given in `dwm\example\dwm-simple\` folder.

2.5 API via BLE interface

This is described in DWM1001-Bletooth-API document.

2.6 API via SPI interface

2.6.1 DWM1001 SPI overview

DWM1001 SPI interface uses TLV format data. Users can use an external host device in SPI master mode to connect to the DWM1001 module SPI interface which operates in slave mode. The maximum SPI clock frequency is 8 MHz. (This is maximum supported by the nRF52 MCU)

In the DWM1001 SPI schemes, host device communicates with the DWM1001 through TLV requests. A full TLV request communication flow includes the following steps:

- 1) Host device sends TLV request;
- 2) DWM1001 prepares response;
- 3) Host device reads the length of total response;
- 4) Host device reads data response;

Because SPI uses full duplex communication, when the host device (as the SPI master) writes x bytes, it actually sends x bytes to the DWM1001 module (as the slave), and receives x bytes dummy in the same time. Similarly for reading, the host device sends x bytes of dummy, and receives x bytes data back as response. In the DWM1001 SPI scheme, the dummy bytes are octets of value `0xFF`.

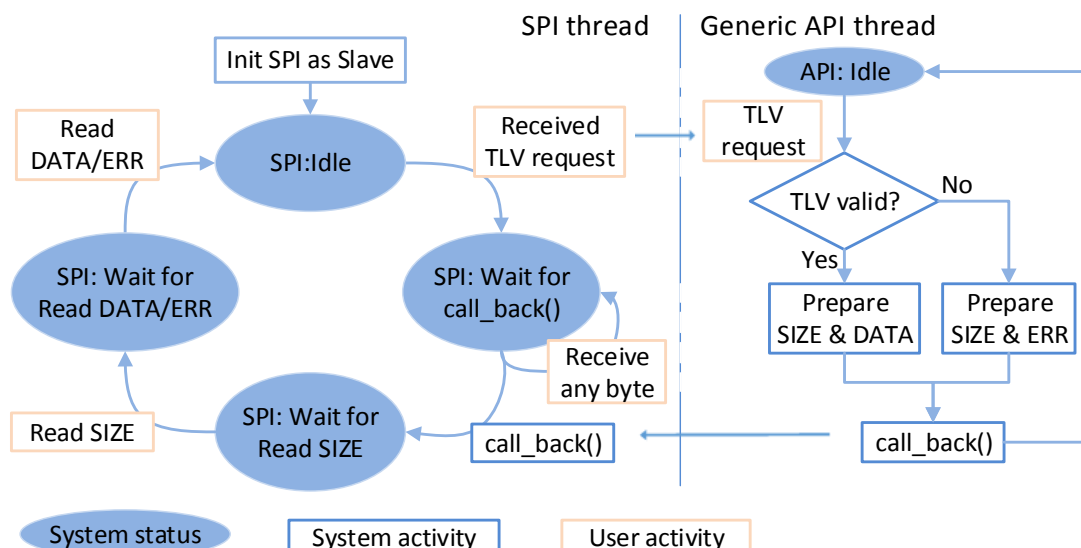


Figure 1 DWM1001 SPI work flow

As shown in Figure 1, the DWM1001 SPI thread transfers between four states in serial. Each state will transfer to its next corresponding state on certain events.

- **SPI: Idle:** the state after initialization and after each successful data transmission/response. In this state, any received data is assumed to be a TLV request. Thus, on receiving any data, the SPI thread in the firmware passes the received data to the Generic API. The Generic API parses the TLV requests and prepares the return data.
 - o Waiting for event: receiving TLV requests.
 - o Action on event – if Type==0xFF:
 - No action, stay in SPI: Idle. See Section 2.6.5 for more details.
 - o Action on event – if Type≠0xFF :
 - Send received TLV request to Generic API thread.
 - Transfer to “SPI: Wait for call_back”.
- **SPI: Wait for call_back:** the DWM1001 SPI is waiting for the Generic API to parse the TLV request and prepare the response. Any data from the host side will be ignored and returned with 0x00 in this state.
 - o Waiting for event: call_back() function being called by the Generic API.
 - o Action on event:
 - Toggle data ready pin HIGH – detailed in Section 2.6.4.
 - Transfer to “SPI: Wait for READ SIZE”.
- **SPI: Wait for Read SIZE:** the DWM1001 SPI has prepared the SIZE byte as response, which is only one byte non-zero data indicating the size of data or error message to be responded, and is waiting for the host device to read the SIZE byte.
 - o Waiting for event: host device reads one byte.
 - o Action on event:
 - Respond with the SIZE byte.
 - Transfer to “SPI: Wait for READ DATA/ERR”.
- **SPI: Wait for Read DATA/ERR:** the DWM1001 SPI has prepared SIZE bytes of data or error message as response to the TLV request, and is waiting for the host device to read it.
 - o Waiting for event: host device reads any number of bytes.

- Action on event:
 - Respond with DATA/ERR.
 - Toggle data ready pin LOW – detailed in Section 2.6.4.
 - Transfer to “SPI: Idle”.

In DWM1001, starting from “SPI: Idle”, traversing all four states listed above and returning to “SPI: Idle” indicates a full TLV request communication flow. The user should have received the response data or error message by the end of the communication flow.

A few different usages and examples are illustrated in the following sub-sections.

2.6.2 SPI Scheme: normal TLV communication

Figure 2 shows the normal communication flow of a host device writing/reading information to/from the DWM1001 module:

- 1) Host device sends the request in TLV format.
- 2) Host device reads the SIZE byte, indicating the number of bytes ready to be read.
 - a. On receiving SIZE= 0, meaning the response is not ready yet, repeat step 2).
 - b. On receiving SIZE > 0, meaning the response is ready, go to step 3)
- 3) Host device reads SIZE bytes data response in TLV format.

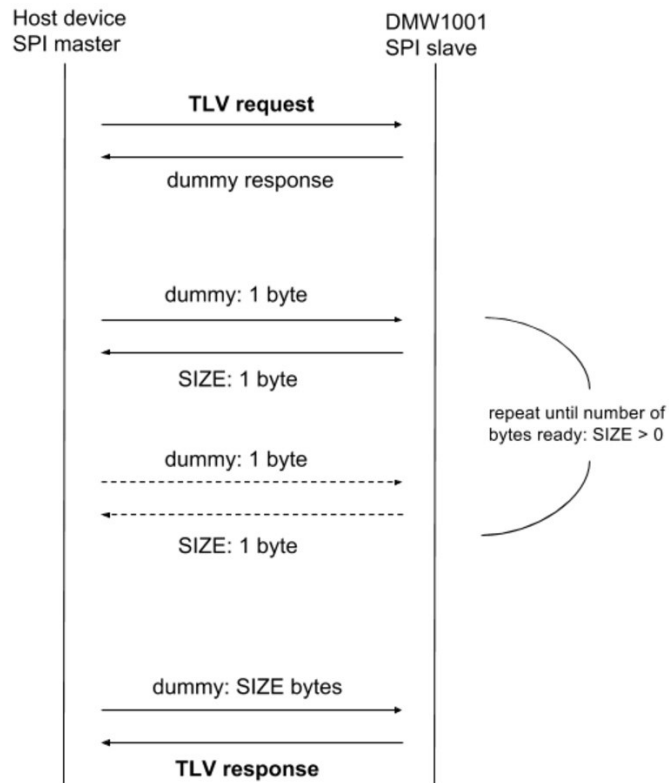


Figure 2 SPI scheme: normal TLV communication

2.6.3 SPI Example: normal TLV communication - dwm_gpio_cfg_output

Figure 3 illustrates an example of host device writing the dwm_gpio_cfg_output TLV request to set pin 13 level HIGH ([0x28, 0x02, 0x0D, 0x01] in byte array, detailed in Section 4.3.14) and reading the response from the DWM1001 module.

The communication flow of the transmission and response includes:

- 1) Host device writes the dwm_gpio_cfg_output command to set pin 13 level HIGH in TLV format, 4 bytes in total:
Type = 0x28, length = 0x02, value = 0x0D 0x01.
- 2) Host device reads the SIZE byte, and receives SIZE = 0: the response is not ready yet. Repeat to read the SIZE byte.
- 3) Host device reads the SIZE byte again, and receives SIZE = 3: the response is ready. Proceed to read 3 bytes response data.
- 4) Host device reads the 3-byte TLV format data:
Type = 0x40, Length = 0x01, value = 0x00,
where Type = 0x40 indicates this is a return message (see Section 6), Length = 0x01 indicates that there is one byte following as data, value = 0x00 indicates the TLV request is parsed correctly.

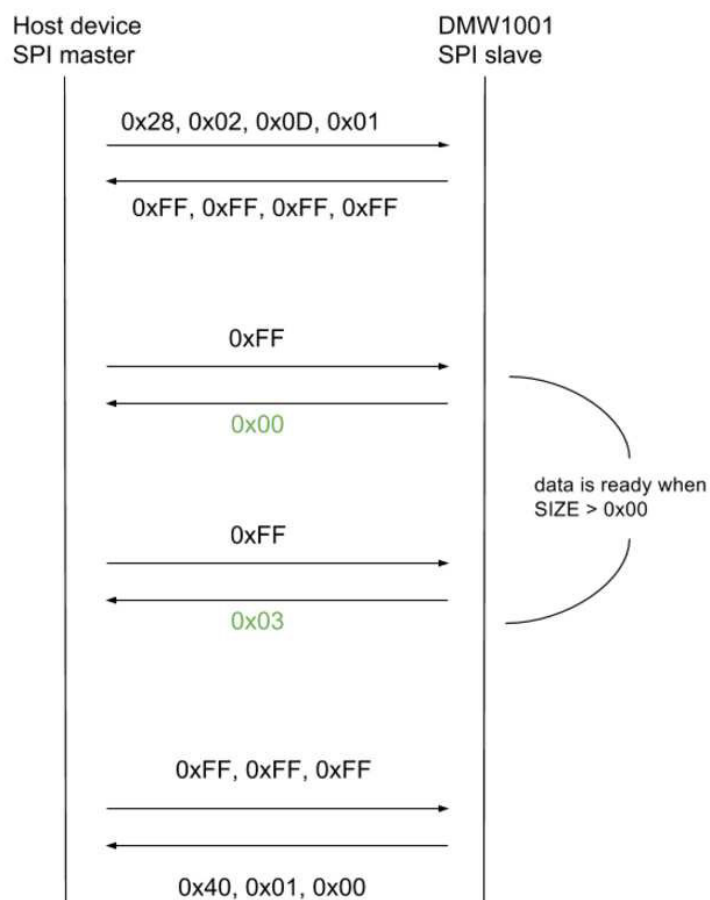


Figure 3 SPI example: normal TLV communication

2.6.4 SPI Scheme: TLV communication using data ready pin

Similar to the scheme described in Section 2.6.2, users can setup the data ready pin (GPIO P0.26) from the DWM1001 to indicate when data is ready, instead of the master polling multiple times to check the response status. When the data ready function is setup, the data ready pin will be set to LOW level when there's no data to be read; when the response SIZE and data is ready to be read, the data ready pin will be set to HIGH level during states "SPI: Wait for Read SIZE" and "SPI: Wait for Read DATA/ERR". Thus, the users can use the data ready pin either as an interrupt or as a status pin.

To setup data ready pin for SPI scheme, users need to use `dwm_int_cfg` TLV request through SPI Scheme: normal TLV communication introduced in Section 2.6.2. The detail of `dwm_int_cfg` request is introduced in Section 4.3.14.

The communication flow of this scheme is illustrated in Figure 4.

- 1) Setup the SPI interrupt.
- 2) Host device writes the request in TLV format.
- 3) Host device waits until the data ready pin on DWM1001 to go HIGH.
- 4) Host device reads the SIZE byte.
- 5) Host device reads SIZE bytes data response in TLV format

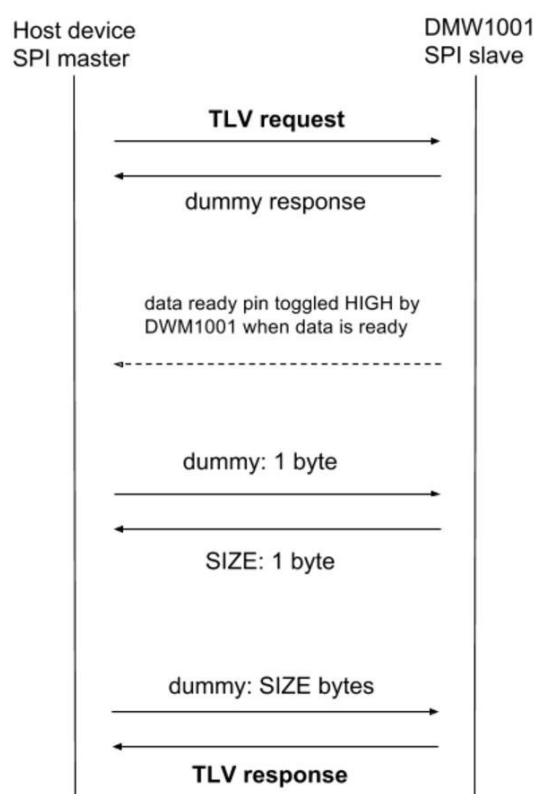


Figure 4 SPI scheme: TLV communication using data ready pin

As can be seen from the steps, this scheme is using the data ready pin on the DWM1001 to indicate the host device when the response data is ready. This makes the host device less busy in reading SIZE byte.

2.6.5 SPI error recovery mechanism

2.6.5.1 SPI data doesn't allow partial transmission

When reading data from the DWM1001 module, if the host device doesn't read all bytes of data in one transmission, the reading operation will still be considered as done. The rest of the response will be abandoned. For example, in "SPI: Wait for Read DATA/ERR" state, the DWM1001 module has prepared SIZE bytes of response data and expects the host device to read all SIZE bytes of the response. However, if the host device only reads part of the data, the DWM1001 module will drop the rest of the data, and transfers to the next state: "SPI: IDLE".

2.6.5.2 SPI state recovery: type_nop message

The DWM1001 SPI has a special Type value 0xFF, called type_nop. A TLV data message with type_nop means no operation. In "SPI: IDLE" state, when the DWM1001 SPI receives a message and finds the type byte is 0xFF, it will not perform any operation, including sending the TLV data message to the Generic API thread.

The type_nop is designed for error recovery. If the host device is not sure what state the DWM1001 SPI is in, it can make use of the SPI response and the non-partial transmission mechanism, and reset the DWM1001 SPI to "SPI: IDLE" state by sending three 0xFF dummy bytes, each in a single transmission. After the three transmissions, the response data from the DWM1001 SPI will become all dummy bytes of value 0xFF, indicating that the DWM1001 SPI is in "SPI: IDLE" state.

2.7 API via UART interface

2.7.1 DWM1001 UART overview

Users can use an external host device to connect to the DWM1001 module through UART interface at baud rate 115200. Figure 5 shows the work flow of the DWM1001 UART interface. In the UART Generic mode communication, the host device is acting as the initiator, while the DWM1001 module is the responder.

DWM1001 UART provides two modes: the UART Generic mode, commands introduced in Section 4, and the UART Shell mode, commands introduced in Section 5. The default mode of the DWM1001 UART is Generic mode. However, the two modes are transferrable:

Generic mode to Shell mode: press “Enter” twice or input two bytes [0x0D, 0x0D] within one second. If the module is in “sleep” state in low power mode as introduced in Section 2.2, an extra byte will be needed before the double “Enter”. E.g. pressing “Enter” three times will wake up the module and transfer it to Shell mode.

Shell mode to Generic mode: users need to input “quit” command.

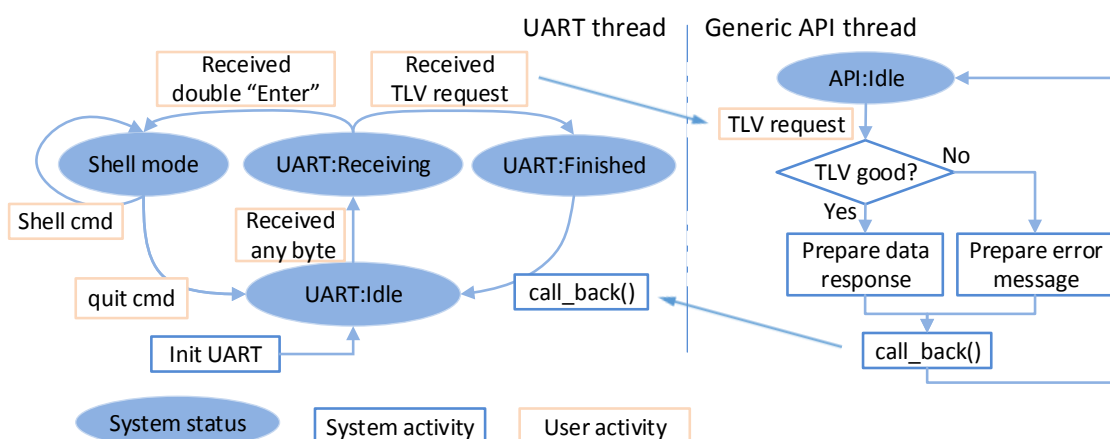


Figure 5 DWM1001 UART work flow

2.7.2 UART TLV Mode

UART TLV mode uses TLV format data. In this mode, host device and the DWM1001 module communicate using TLV requests/responses. A full TLV request communication flow includes the following steps:

- 1) Host device sends TLV request;
- 2) DWM1001 responds with TLV data.

On receiving any data, the UART starts a delay timer for the following data. If there is new data coming in within a delay period, specifically 25 clock cycles (25/32768 second ≈ 763 μs), the UART starts the delay timer and waits for new data. If there’s no new data coming in within the delay period, the delay timer will expire. The UART then sends the received data to the Generic API thread and waits for it to return the response data or error message.

As shown in Figure 5, the DWM1001 UART TLV mode thread transfers between three states in serial: “UART: Idle”, “UART: Receiving” and “UART: Finished”. Each state will transfer to its next corresponding state on certain events.

- **UART: Idle:** is the state after initialization and after each successful TLV response. In this state, the UART is only expecting one byte as the start of the TLV request or the double “Enter” command.
 - Waiting for event: receiving TLV requests.
 - Action on event:
 - Start the delay timer.
 - Transfer to UART: Receiving.
- **UART: Receiving:** is the state waiting for end of the incoming request. On receiving any data in this state, the UART will refresh the delay timer. If the host device has finished sending bytes, the delay timer will expire.
 - Waiting for event: delay period timed out.
 - Action on event - if received request is double “Enter”:
 - Transfer to UART Shell mode.
 - Action on event - if received request is not double “Enter”:
 - Send received request to Generic API thread.
 - Transfer to UART: Finished.
- **UART: Finished:** is the state waiting for the Generic API thread to parse the incoming request and send the response data or error message back to UART thread.
 - Waiting for event: call_back() function called by the Generic API thread.
 - Action on event:
 - Send the response data or error message to host device.
 - Transfer to UART: Idle.

2.7.3 UART scheme: TLV mode communication

The UART communication in TLV mode is illustrated in Figure 6, steps described as below:

- 1) The host device sends a request in TLV format;
- 2) The DWM1001 module responds with a message in TLV format.

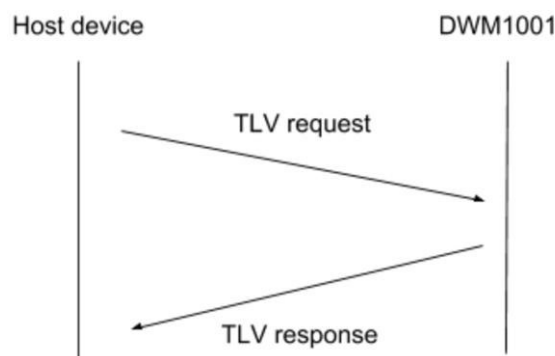


Figure 6 UART scheme: TLV mode communication

2.7.4 UART example: TLV mode communication

Figure 7 illustrates an example of host device sending the `dwm_gpio_cfg_output` command to set pin 13 level HIGH ([0x28, 0x02, 0x0D, 0x01] in byte array, detailed in Section 4.3.14) and receiving the response from the DWM1001 module using the UART API in TLV mode. The steps of the communication flow are:

- 1) The host device sends the `dwm_gpio_cfg_output` command in TLV format:
Type = 0x28, Length = 0x02, Value = 0x0D 0x01.
- 2) The DWM1001 module responds in TLV format:
Type = 0x40, Length = 0x01, value = 0x00,
where Type = 0x40 indicates this is a return message (see Section 6), Length = 0x01 indicates that there is one byte following as data, value = 0x00 indicates the TLV request is parsed correctly.

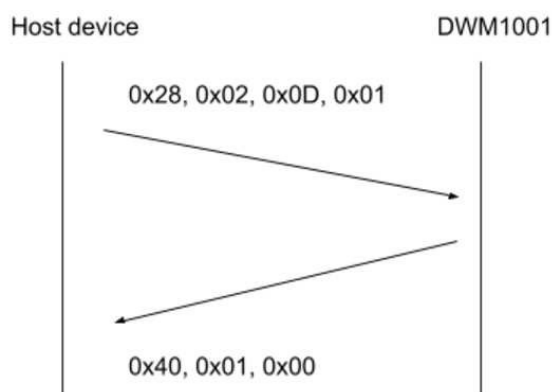


Figure 7 UART example: TLV mode communication

2.7.5 UART scheme: Shell mode communication

UART Shell mode provides prompt and uses Shell commands. The UART Shell mode intends to provide users with human readable access to the APIs, thus, all Shell commands are strings of letters followed by a character return, i.e. “Enter”. Users can input the string directly through keyboard and press “Enter” to send the Shell commands. DWM1001 UART stays in the Shell mode after each Shell commands except for the “quit” command.

As illustrated in Figure 8, a full Shell command communication flow includes the following steps:

- 1) Host device sends the Shell command + “Enter” to the DWM1001.
- 2) If there’s any message to respond, the DWM1001 sends the message to the host device.
- 3) If there’s nothing to respond, the DWM1001 doesn’t send anything and keeps quiet.

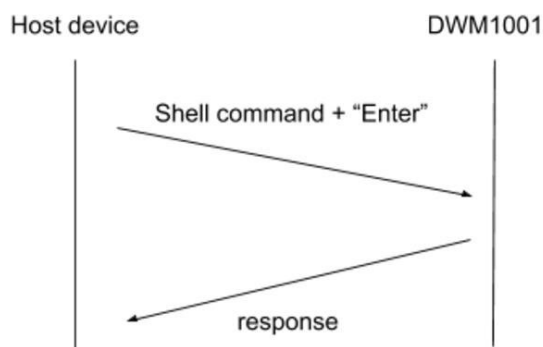


Figure 8 UART scheme: Shell mode communication

2.7.6 UART example: Shell Mode communication

Figure 9 illustrates an example of host device sending the “GPIO set” command using UART Shell to set pin 13 level HIGH (“gs 13” in byte array, followed by “Enter” key, detailed in Section 5.7). The steps of the communication flow are:

- 1) The host device sends the “GPIO set” command in Shell mode: “gs 13” + “Enter”.
- 2) The DWM1001 responds the host with string “gpio13: 1”.

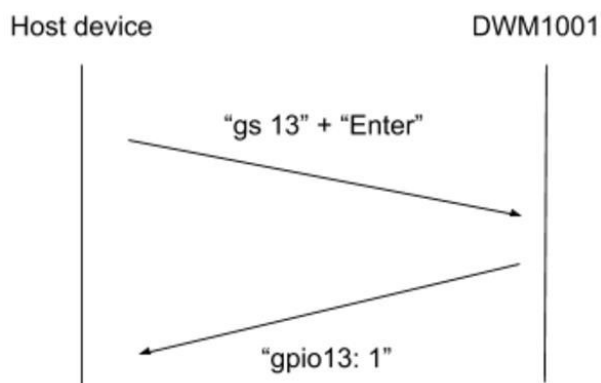


Figure 9 UART example: Shell mode communication

2.8 GPIO Scheme: DWM1001 notifies for status change

Rather than the host device initiating the SPI/UART communication, the DWM1001 module can send notifications of status change by toggling the dedicated GPIO pin (P0.26) to HIGH level, as illustrated in Figure 10. To enable this feature, host device needs to use the `dwm_int_cfg` command, detailed in Section 4.3.14. On detecting the HIGH level, host device can initiate a `dwm_status_get` command, detailed in Section 0, to get the status from the DWM1001 device. Both `dwm_int_cfg` and `dwm_status_get` commands can be sent through SPI or UART schemes introduced in the previous sections.

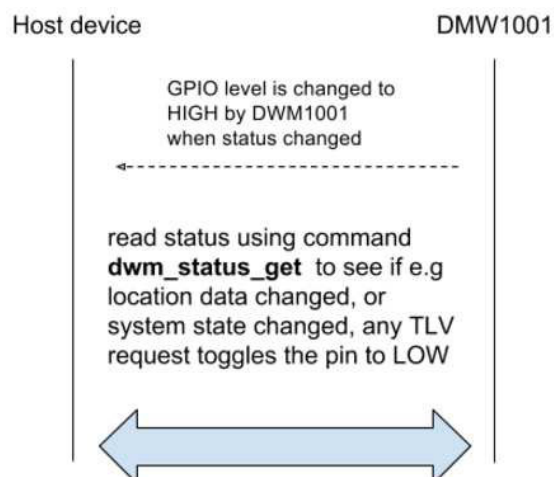


Figure 10 GPIO scheme: DWM1001 notifies host device of status change, using GPIO

This GPIO pin level change will be postponed if the status change happens during the SPI TLV request/response procedure described above to avoid conflict. In detail, when the SPI is in status: “SPI: Wait for call_back”, “SPI: Wait for Read SIZE” and “SPI: Wait for Read DATA/ERR”, the GPIO scheme will surrender the control of the GPIO pin. After the SPI communication when the SPI is in status “SPI: Idle”, the GPIO scheme will regain the control of the GPIO pin.

2.9 API for on-board C code developers

Decawave provides the DWM1001 source pack of the pre-built firmware. Users can add their own code and make use of the C code API functions in certain entry files provided in the source pack. In this way, users are able to add their own functions inside the module firmware and perhaps not need to add an external host controller device. The detail of adding such user code is provided in DWM1001 Firmware User Guide [3].

A few points the C code users should note when using the on-board firmware:

- 1) User application is based on eCos RTOS and DWM libraries.
- 2) Files used for linking with the user applications:
 - a. dwm.h - header file - wrapper for all header files necessary for building of the user application
 - b. libdwm.a - static library
 - c. extras.o, vectors.o, libtarget.a - eCos static library
 - d. target_s132_fw1.ld - linker script for firmware section 1
 - e. target_s132_fw2.ld - linker script for firmware section 2
- 3) The API provides functions and defines to the user application
 - a. Common functions on operating systems like thread creation, memory allocation, access to interfaces (e.g. GPIO, SPI), synchronization (e.g. mutex, signal)
 - b. Initialization, configuration and maintenance of the DWM communication stack
 - c. Register of callbacks for incoming data and measurements

3 GENERIC API INFORMATION

3.1 *Used terminology*

Anchor: has a fixed location – as a reference point to locate Tags. The module may be configured to behave as an anchor node. An anchor initiator is an anchor with the initiator flag enabled as introduced in section 3.4.8. It is a special anchor node that initiates the whole network, see the system overview for more detail [4].

Tag: usually moving/changing location, determines its position dynamically with the help of the anchors. The module may be configured to behave as a tag node.

Gateway: knows about all nodes in the network, provides status information about network nodes (read/inspect), cache this information and provide it to gateway client, provides means to interact with network elements (a.k.a. interaction proxy).

Node: network node (anchor, tag, gateway...)

LE: location engine – position solver function (on the tag)

3.2 *Little endian*

The integers used in the PANS API are little endian, unless otherwise stated.

3.3 *Firmware update*

As introduced in the DWM1001 System Overview [4], the nodes will compare their firmware version to the network they want to join. If the firmware version is different, the nodes will try to update their firmware before joining. This firmware update function can be enabled/disabled in the configuration. Here lists the rules of the function that the nodes will follow.

Tag:

When enabled, the tag will always check the firmware version and try to synchronise its firmware version with the network by sending the update request to the nearby anchor nodes in the network before it starts ranging.

When disabled, the tag will start ranging without checking the firmware version. This can lead to version compatibility problems and must be dealt with very carefully.

Anchor:

When enabled, before joining the network, the anchor will check the firmware version and try to synchronise its firmware version with the network by sending the update request to the nearby anchor nodes. After having joined the network, the anchor will respond to nearby nodes' requests to update their firmware.

When disabled, before joining the network, the anchor will directly send the join request and will not check the firmware version. This can lead to version compatibility problems and must be dealt

with very carefully. After having joined the network, the anchor will ignore the firmware update requests from the nearby nodes.

3.4 Frequently used TLV values

Below lists the data that are frequently used in the APIs, either as input parameters or output parameters. These parameters are of fixed size and some have their own value ranges.

3.4.1 err_code

1-byte error code, response information to the incoming requests.

err_code : 8-bit integer, valid values:
0 : OK
1 : unknown command or broken TLV frame
2 : internal error
3 : invalid parameter
4 : busy

3.4.2 position

13-byte position information of the node (anchor or tag).

position = x, y, z, qf : bytes 0-12, position coordinates and quality factor
x : bytes 0-3, 32-bit integer, in millimeters
y : bytes 4-7, 32-bit integer, in millimeters
z : bytes 8-11, 32-bit integer, in millimeters
qf : bytes 12, 8-bit integer, position quality factor in percent

3.4.3 gpio_idx

1-byte index of GPIO pins available to the user.

gpio_idx : 8-bit integer, valid values: 2, 8, 9, 10, 12, 13, 14, 15, 22, 23, 27, 30, 31.

Note: some GPIO pins are occupied by the PANS library and can only be accessed/controlled by through the APIs partially:

- When configured as tag and BLE function is enabled, GPIO2 is occupied.
- When LED function is enabled, GPIO 22, 30 and 31 are occupied.
- During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

3.4.4 gpio_value

1-byte value of the GPIO pin.

gpio_value = 8-bit integer, valid values:

0 : set the I/O port pin to a LOW voltage, logic 0 value
1 : set the I/O port pin to a HIGH voltage, logic 1 value

3.4.5 gpio_pull

1-byte status of the GPIO pin as input.

gpio_pull = 8-bit integer, valid values:
0 : DWM_GPIO_PIN_NOPULL
1 : DWM_GPIO_PIN_PULLDOWN
3 : DWM_GPIO_PIN_PULLUP

3.4.6 fw_version

4-byte value representing the version number of the firmware.

fw_version = **maj**, **min**, **patch**, **ver**: bytes 0-3, firmware version
maj : byte 0, 8-bit number, MAJOR
min : byte 1, 8-bit number, MINOR
patch : byte 2, 8-bit number, PATCH
ver : byte 3, 8-bit number, **res** and **var**
res : byte 3, bits 4-7, 4-bit number, RESERVED
var : byte 3, bits 0-3, 4-bit number, VARIANT

3.4.7 cfg_tag

2 bytes, configuration of the tag, which contains common tag configuration parameters. Each bit represents different thing.

cfg_tag = **accel_en**, **meas_mode**, **low_power_en**, **loc_engine_en**, **ble_en**, **uwb_mode**, **fw_upd_en**: tag configuration information.
accel_en: accelerometer enable.
meas_mode: measurement mode. 0 - TWR; 1, 2, 3 - reserved.
low_power_en: low-power mode enable.
loc_engine_en: internal Location Engine enable. 0 means do not use internal Location Engine, 1 means internal Location Engine.
ble_en: Bluetooth enable.
uwb_mode: UWB operation mode: 0 - offline, 1 – passive, 2 – active.
fw_upd_en: Firmware update enable.

3.4.8 cfg_anchor

1 byte, configuration of the anchor, which contains common anchor configuration parameters. Each bit represents different thing.

cfg_anchor = initiator, bridge, ble_en, uwb_mode, fw_upd_en

initiator : Initiator role enable.

bridge : Bridge role enable.

ble_en : Bluetooth enable.

uwb_mode: UWB operation mode: 0 - offline, 1 – passive, 2 – active.

fw_upd_en : Firmware update enable.

3.4.9 cfg_node

2 bytes, configuration of the node, which contains common configuration parameters. Each bit represents different thing.

cfg_anchor = err_code, mode, initiator, bridge, low_power_en, meas_mode, loc_engine_en, ble_en, uwb_mode, fw_update_en

mode: 0 - tag, 1 - anchor.

initiator : Initiator role enable.

bridge : Bridge role enable.

accel_en: accelerometer enable.

meas_mode: measurement mode. 0 - TWR; 1, 2, 3 - reserved.

low_power_en: low-power mode enable.

loc_engine_en: internal Location Engine enable. 0 means do not use internal Location Engine, 1 means internal Location Engine is active.

ble_en: Bluetooth enable.

uwb_mode: UWB operation mode: 0 - offline, 1 – passive, 2 – active.

fw_upd_en: Firmware update enable.

4 API FUNCTION DESCRIPTIONS

4.1 List of API functions

The API functions for on-board user app (c code) and SPI/UART TLV commands are unified. Listed below in Table 2, in TLV type ascending order.

Table 2 API request function list

API function name	On-board user app available	SPI/UART TLV		Ref (Section)
		type	Length	
dwm_pos_set	y	0x01	0x0d	4.3.1
dwm_pos_get	y	0x02	0x00	4.3.2
dwm_upd_rate_set	y	0x03	0x02	4.3.3
dwm_upd_rate_get	y	0x04	0x00	4.3.4
dwm_cfg_tag_set	y	0x05	0x02	4.3.5
dwm_cfg_anchor_set	y	0x07	0x01	4.3.6
dwm_cfg_get	y	0x08	0x00	4.3.7
dwm_sleep	y	0x0a	0x00	4.3.8
dwm_loc_get	y	0x0c	0x00	4.3.9
dwm_baddr_set	y	0x0f	0x00	4.3.10
dwm_baddr_get	y	0x10	0x00	4.3.11
dwm_reset	y	0x14	0x00	4.3.12
dwm_ver_get	y	0x15	0x00	4.3.13
dwm_gpio_cfg_output	y	0x28	0x02	4.3.14
dwm_gpio_cfg_input	y	0x29	0x02	4.3.15
dwm_gpio_value_set	y	0x2a	0x02	4.3.16
dwm_gpio_value_get	y	0x2b	0x01	4.3.17
dwm_gpio_value_toggle	y	0x2c	0x01	4.3.18
dwm_status_get	n	0x32	0x00	4.3.19
dwm_int_cfg	n	0x34	0x02	4.3.20
dwm_gpio_irq_cfg	y	n/a	n/a	4.3.21
dwm_gpio_irq_dis	y	n/a	n/a	4.3.22
dwm_i2c_read	y	n/a	n/a	4.3.23
dwm_i2c_write	y	n/a	n/a	4.3.24
dwm_evt_cb_register	y	n/a	n/a	4.3.25

The definition and usage of the API functions are described below in individual sub-sections. In introducing each API function, all possible accesses are described, including c code, UART/SPI TLV.

Note: Some API functions are only provided in limited accesses.

Note: There are more TLV type than listed in Table 2. The TLV types that do not relate to API functions are not introduced here, e.g. the TLV response types. See Section 6 for the full TLV type list.

4.2 Usage of the APIs

Examples of how the UART Generic, SPI and C code API can be used is introduced in the DWM1001 Firmware User Guide [3]. The examples intend to give a brief view of the API usage and only include very simple API functions. The code in the source files of the examples can be changed to modify/add functionalities. The full list of API functions as shown in Table 2 can also be found in the included header files of the API source code packages. Table 3 lists the packages with examples of the corresponding APIs, the locations of the header files and the locations of the source files.

Table 3 API examples location

API	Package with examples	Header file location	Example source file location
C code	DWM1001 on-board package [5]	dwm\include\dwm.h	dwm\examples\dwm-simple\dwm-simple.c
UART Generic	DWM1001 Host API package [6]	include\dwm_api.h	examples\ex1_TWR_2Hosts\tag\tag_cfg.c
SPI	DWM1001 Host API package [6]	include\dwm_api.h	examples\ex1_TWR_2Hosts\tag\tag_cfg.c

4.3 Details of the API functions

The details of each API function listed in Table 2 are described in the below sub-sections.

4.3.1 dwm_pos_set

4.3.1.1 Description

This API function set the default position of the node. Default position is not used when in tag mode but is stored anyway so the module can be configured to anchor mode and use the value previously set by dwm_pos_set. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	Name	Size	
Input	position	13-byte array	Position information, see Section 3.4.2
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.1.2 C code

Declaration:

```
int dwm_pos_set(dwm_pos_t* p_pos);
```

Example:

```
dwm_pos_t pos;
pos.qf = 100;
pos.x = 121;
pos.y = 50;
pos.z = 251;
dwm_pos_set(&pos);
```

4.3.1.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
0x01	0x0D	position

Example:

TLV request					
Type	Length	Value			
		Position			
		32 bit value in little endian is x coordinate in millimeters	32 bit value in little endian is y coordinate in millimeters	32 bit value in little endian is z coordinate in millimeters	8 bit value is quality factor in percents (0-100)
0x01	0x0D	0x79 0x00 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0x00 0xfb 0x00 0x00 0x00 0x64			

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.2 dwm_pos_get

4.3.2.1 Description

This API function obtain position of the node. If the current position of the node is not available, the default position previously set by dwm_pos_set will be returned.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1
	position	13-byte array	position information, see Section 3.4.2

4.3.2.2 C code

Declaration:

```
int dwm_pos_set(dwm_pos_t* p_pos);
```

Example:

```
dwm_pos_t pos;
dwm_pos_get(&pos);
```

4.3.2.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x02	0x00

Example:

TLV request	
Type	Length
0x02	0x00

TLV response								
Type	Length	Value	Type	Length	Value			
		err_code			Position			
					32 bit value in little endian is x coordinate in millimeters	32 bit value in little endian is y coordinate in millimeters	32 bit value in little endian is z coordinate in millimeters	8 bit value is quality factor in percents (0-100)
0x40	0x01	0x00	0x41	0x0D	0x79 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0xfb 0x00 0x00 0x00 0x64			

4.3.3 dwm_upd_rate_set

4.3.3.1 Description

This API function sets the update rate and the stationary update rate of the position in unit of 100 milliseconds. Stationary update rate must be greater or equal to normal update rate. This call does a write to the internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	Name	Size	
Input	update_rate	16-bit integer	position publication interval in multiples of 100 milliseconds
	update_rate_stationary	16-bit integer	position publication interval when node is not moving in multiples of 100 milliseconds, maximum is 2 minutes, must be greater or equal to normal update_rate
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.3.2 C code

Declaration:

```
int dwm_upd_rate_set(uint16_t ur, uint16_t urs);
```

Example:

```
dwm_upd_rate_set(10, 50); // update rate 1 second. 5 seconds stationary
```

4.3.3.3 SPI/UART Generic

Declaration:

TLV request			
Type	Length	Value	
0x03	0x04	update_rate	update_rate_stationary

Example:

TLV request			
Type	Length	Value	
		update_rate	update_rate_stationary
		16 bit value in little endian, which is update rate in multiples of 100 ms (e.g. 0x0A 0x00 means 10)	16 bit value in little endian, which is stationary update rate in multiples of 100 ms. E.g. 0x0A 0x00 means 5.0.
0x03	0x04	0x0A 0x00	0x32 0x00

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.4 dwm_upd_rate_get

4.3.4.1 Description

This API function gets position update rate.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1
	update_rate	16-bit integer	position update interval in multiples of 100 milliseconds,
	update_rate_stationary	16-bit integer	Stationary position update interval in multiples of 100 milliseconds

4.3.4.2 C code

Declaration:

```
int dwm_upd_rate_get(uint16_t* p_ur, uint16_t* p_urs);
```

Example:

```
uint16_t ur, urs;
dwm_upd_rate_get(&ur, &urs);
```

4.3.4.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x04	0x00

Example:

TLV request	
Type	Length
0x04	0x00

TLV response						
Type	Length	Value	Type	Length	Value	
		err_code			update_rate, first 2 bytes, indicating the position update interval in multiples of 100 ms. E.g. 0x0A 0x00 means 1.0s interval.	update_rate_stationary, second 2 bytes, indicating the stationary position update interval in multiples of 100 ms. E.g. 0x32 0x00 means 5.0s interval.
0x40	0x01	0x00	0x45	0x04	0x0A 0x00	0x32 0x00

4.3.5 dwm_cfg_tag_set

4.3.5.1 Description

This API function configures the node as tag with given options. It automatically resets the DWM module. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds. Note that this function only sets the configuration parameters. To make effect of the settings, users should issue a reset command, see section 4.3.12 for more detail.

Parameter			Description
Field	Name	Size	
Input	cfg_tag	16-bit integer	2-byte, configuration of the tag, see Section 3.4.7
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.5.2 C code

Declaration:

```
int dwm_cfg_tag_set(dwm_cfg_tag_t* p_cfg);
```

Example:

```
dwm_cfg_tag_t cfg;

cfg.accel_en = 0;
cfg.low_power_en = 1;
cfg.meas_mode = DWM_MEAS_MODE_TWR;
cfg.loc_engine_en = 1;
cfg.common.ble_en = 1;
cfg.common.online = 1;
cfg.common.fw_update_en = 1;
cfg.common.uwb_mode = DWM_UWB_MODE_ACTIVE;
dwm_cfg_tag_set(&cfg);
```

4.3.5.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
		(* BYTE 1 *) (bits 3-7) reserved (bit 2) accel_en (bits 0-1) meas_mode : 0 - TWR, 1-3 reserved
		(* BYTE 0 *) (bit 7) low_power_en (bit 6) loc_engine_en (bit 5) reserved (bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en (bits 0-1) uwb_mode
0x05	0x02	cfg_tag

Example:

TLV request		
Type	Length	Value
		cfg_tag: low_power_en, loc_engine_en, ble_en, DWM_UWB_MODE_ACTIVE, fw_update_en
0x03	0x04	0xCC 0x00

TLV response

Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.6 dwm_cfg_anchor_set

4.3.6.1 Description

This API function configures the node as tag with given options. It automatically resets the DWM module. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds. Note that this function only sets the configuration parameters. To make effect of the settings, users should issue a reset command, see section 4.3.12 for more detail.

Parameter			Description
Field	Name	Size	
Input	cfg_anchor	8-bit integer	1-byte, configuration of the tag, see Section 3.4.8
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.6.2 C code

Declaration:

```
int dwm_cfg_anchor_set(dwm_cfg_anchor_t* p_cfg)
```

Example:

```
dwm_cfg_anchor_t cfg;

cfg.initiator = 1;
cfg.bridge = 0;
cfg.common.ble_en = 1;
cfg.common.online = 1;
cfg.common.fw_update_en = 1;
cfg.common.uwb_mode = DWM_UWB_MODE_ACTIVE;
dwm_cfg_anchor_set(&cfg);
```

4.3.6.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
		(bit 7) initiator (bit 6) bridge (bit 5) reserved (bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en (bits 0-1) uwb_mode
0x07	0x01	cfg_anchor

Example:

TLV request		
Type	Length	Value
		cfg_anchor: initiator, ble_en, fw_update_en, UWB_MODE_ACTIVE
0x03	0x04	0x8E

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.7 dwm_cfg_get

4.3.7.1 Description

This API function obtains the configuration of the node.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1
	cfg_node	16-bit integer	configuration of the node, see Section 3.4.9

4.3.7.2 C code

Declaration:

```
int dwm_cfg_get(dwm_cfg_t* p_cfg);
```

Example:

```
dwm_cfg_t cfg;

dwm_cfg_get(&cfg);

printf("mode %u \n", cfg.mode);
printf("initiator %u \n", cfg.initiator);
printf("bridge %u \n", cfg.bridge);
printf("motion detection enabled %u \n", cfg.accel_en);
printf("measurement mode %u \n", cfg.meas_mode);
printf("low power enabled %u \n", cfg.low_power_en);
printf("internal location engine enabled %u \n", cfg.loc_engine_en);
printf("LED enabled %u \n", cfg.common.led_en);
printf("BLE enabled %u \n", cfg.common.ble_en);
printf("firmware update enabled %u \n", cfg.common.fw_update_en);
printf("UWB mode %u \n", cfg.common.uwb_mode);
```

4.3.7.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x08	0x00

Example:

TLV request	
Type	Length
0x08	0x00

TLV response					
Type	Length	Value	Type	Length	Value
		err_code			cfg_node: (* BYTE 1 *) (bits 6-7) reserved (bit 5) mode : 0 - tag, 1 - anchor (bit 4) initiator (bit 3) bridge (bit 2) accel_en (bits 0-1) meas_mode : 0 - TWR, 1-3 not supported (* BYTE 0 *) (bit 7) low_power_en (bit 6) loc_engine_en (bit 5) reserved (bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en

					(bits 0-1) uwb_mode
0x40	0x01	0x00	0x46	0x02	0x07 0x04

4.3.8 dwm_sleep

4.3.8.1 Description

This API function puts the module into sleep mode. Low power option must be enabled otherwise an error will be returned.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.8.2 C code

Declaration:

```
int dwm_sleep(void);
```

Example:

```
dwm_sleep();
```

4.3.8.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x0A	0x00

Example:

TLV request	
Type	Length
0x0A	0x00

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.9 dwm_loc_get

4.3.9.1 Description

Get last distances to the anchors (tag is currently ranging to) and the associated position. The interrupt is triggered when all TWR measurements have completed and the LE has finished. If the LE is disabled, the distances will just be returned. This API works the same way in both Responsive and Low-Power tag modes.

Parameter			Description
Field	Name	Size	
Input	none		
Output	position	13-byte array	Position information of the current node, see Section 3.4.2
	dist.cnt	1-byte	Number of distances to the anchors, max 15.
	dist.addr	8-byte/2-byte	public UWB address in little endian. ⁽¹⁾
	dist.dist	4-byte	Distances to the anchors. ⁽¹⁾
	dist.qf	1-byte	Quality factor of distances to the anchors. ⁽¹⁾
	an_pos.cnt	1-byte	Number of anchor positions, max 15.
	an_pos	13-byte array	Anchor positions information, see Section 3.4.2. ⁽²⁾

(1) This data can appear more than once according to the value of dist.cnt. This data is 8-byte long in C code API, 8-byte long in SPI/UART response for Anchor, and 2-byte long for SPI/UART response for Tag.

(2) This data can appear more than once according to the value of an_pos.cnt.

4.3.9.2 C code

Declaration:

```
int dwm_loc_get(dwm_loc_data_t* p_loc);
```

Example:

```
dwm_loc_data_t loc;
dwm_pos_t pos;
int rv, i;

loc.p_pos = &pos;
rv = dwm_loc_get(&loc);

if (0 == rv) {
    PRINT("[%ld,%ld,%ld,%u] ", loc.p_pos->x, loc.p_pos->y, loc.p_pos->z,
        loc.p_pos->qf);

    for (i = 0; i < loc.anchors.dist.cnt; ++i) {
        PRINT("%u", i);
        PRINT("0x%llx", loc.anchors.dist.addr[i]);
        if (i < loc.anchors.an_pos.cnt) {
            PRINT("[%ld,%ld,%ld,%u]", loc.anchors.an_pos.pos[i].x,
                loc.anchors.an_pos.pos[i].y,
                loc.anchors.an_pos.pos[i].z,
                loc.anchors.an_pos.pos[i].qf);
        }

        PRINT("=%lu,%u ", loc.anchors.dist.dist[i], loc.anchors.dist.qf[i]);
    }
    PRINT("\n");
} else {
    PRINT("err code: %d\n", rv);
}
```

4.3.9.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length

0x0C	0x00
------	------

Example 1 (Tag node):

TLV request	
Type	Length
0x0C	0x00

TLV response												
Type	Length	Value	Type	Length	Value				Type	Length	Value	
		err_code			position, 13 bytes						1 byte, number of distances encoded in the value	
					4-byte x	4-byte y	4-byte z	1-byte quality factor				
0x40	0x01	0x00	0x41	0x0D	0x4b	0x0a	0x00	0x00	0x1f	0x49	0x51	0x04
					0x04	0x00	0x00	0x9c	0x0e			
					0x00	0x00	0x64					

TLV response (residue of the frame from previous table)									
Value									
2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format	...	2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format	
position and distance AN1				AN2, AN3	position and distance AN4				

Example 2 (Anchor node):

TLV request	
Type	Length
0x0C	0x00

TLV response												
Type	Length	Value	Type	Length	Value				Type	Length	Value	
		err_code			position, 13 bytes						1 byte, number of distances encoded in the value	
					4-byte x	4-byte y	4-byte z	1-byte quality factor				
0x40	0x01	0x00	0x41	0x0D	0x4b	0x0a	0x00	0x00	0x1f	0x48	0xC4	0x0F
					0x04	0x00	0x00	0x9c	0x0e			
					0x00	0x00	0x64					

TLV response (residue of the frame from previous table)							
Value							
8 bytes UWB address	4-byte distance	1-byte distance quality factor	...	8 bytes UWB address	4-byte distance	1-byte distance quality factor	
distance AN1			AN2, ..., AN14	distance AN15			

4.3.10 dwm_baddr_set

4.3.10.1 Description

Sets the public Bluetooth address used by device. New address takes effect after reset. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds

Parameter			Description
Field	Name	Size	
Input	ble_addr	6-bytes	48-bit long public BluetoothE address in little endian.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.10.2 C code

Declaration:

```
int dwm_baddr_set(dwm_baddr_t* p_baddr);
```

Example:

```
dwm_baddr_t baddr;

baddr.byte[0] = 1;
baddr.byte[1] = 2;
baddr.byte[2] = 3;
baddr.byte[3] = 4;
baddr.byte[4] = 5;
baddr.byte[5] = 6;
dwm_baddr_set(&baddr);
```

4.3.10.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
0x0F	0x06	ble_addr

Example:

TLV request		
Type	Length	Value
		ble_addr , 6 bytes in little endian
0x0F	0x06	0x01 0x23 0x45 0x67 0x89 0xab

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.11 dwm_baddr_get

4.3.11.1 Description

Get Bluetooth address currently used by device.

Parameter			Description
Field	Name	Size	
Input	none		
Output	ble_addr	6-bytes	48-bit long public Bluetooth address in little endian.

4.3.11.2 C code

Declaration:

```
int dwm_baddr_get(dwm_baddr_t* p_baddr);
```

Example:

```
dwm_baddr_t baddr;
int i;

if (DWM_OK == dwm_baddr_get(&baddr)) {
    printf("addr=");
    for (i = DWM_BLE_ADDR_LEN - 1; i >= 0; --i) {
        printf("%02x%s", baddr.byte[i], (i > 0) ? " : " : "");
    }
    printf("\n");
} else {
    printf("FAILED");
}
```

4.3.11.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x10	0x00

Example:

TLV request	
Type	Length
0x10	0x00

TLV response					
Type	Length	Value	Type	Length	Value
		err_code			ble_addr, 6 bytes in little endian
0x40	0x01	0x00	0x5F	0x06	0x01 0x23 0x45 0x67 0x89 0xab

4.3.12 dwm_reset

4.3.12.1 Description

This API function reboots the module.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.12.2 C code

Declaration:

```
int dwm_reset(void);
```

Example:

```
dwm_reset();
```

4.3.12.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x14	0x00

Example:

TLV request	
Type	Length
0x14	0x00

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.13 dwm_ver_get

4.3.13.1 Description

This API function obtains the firmware version of the module.

Parameter			Description
Field	Name	Size	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1
	fw_version	4-byte array	firmware version, see Section 3.4.6
	cfg_version	4-byte array	configuration version, 32-bits integer
	hw_version	4-byte array	hardware version, 32-bits integer

4.3.13.2 C code

Declaration:

```
int dwm_ver_get(dwm_ver_t* p_ver);
```

Example:

```
dwm_ver_t ver;
dwm_ver_get(&ver);
```

4.3.13.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x15	0x00

Example:

TLV request	
Type	Length
0x15	0x00

TLV response								
Type	Length	Value	Type	Length	Value	Type	Length	Value
		err_code			fw_version: maj = 1 min = 2 patch = 5 var = 1			cfg_version
0x40	0x01	0x00	0x50	0x04	0x01 0x05 0x02 0x01	0x51	0x04	0x00 0x07 0x01 0x00

TLV response (residue of the frame from previous table)		
Type	Length	Value
		hw_version
0x52	0x04	0x2a 0x00 0xca 0xde

4.3.14 dwm_gpio_cfg_output

4.3.14.1 Description

This API function configures a specified GPIO pin as an output and also sets its value to 1 or 0, giving a high or low digital logic output value.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values
	gpio_value	8-bit integer	0 or 1
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.14.2 C code

Declaration:

```
int dwm_gpio_cfg_output(dwm_gpio_idx_t idx, bool value);
```

Example:

```
dwm_gpio_cfg_output(DWM_GPIO_IDX_13, 1); // set pin 13 as output and to 1 (high voltage)
```

4.3.14.3 SPI/UART Generic

Declaration:

TLV request			
Type	Length	Value	
0x28	0x02	gpio_idx	gpio_value

Example:

TLV request			
Type	Length	Value	
		gpio_idx	gpio_value
0x28	0x02	0x0d	0x01

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.15 dwm_gpio_cfg_input

4.3.15.1 Description

This API function configure GPIO pin as input.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
	gpio_pull	8-bit integer	GPIO pull status, see Section 3.4.53.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.15.2 C code

Declaration:

```
int dwm_gpio_cfg_input(dwm_gpio_idx_t idx, dwm_gpio_pin_pull_t pull_mode);
```

Example:

```
dwm_gpio_cfg_input(DWM_GPIO_IDX_13, DWM_GPIO_PIN_PULLUP);
```

4.3.15.3 SPI/UART Generic

Declaration:

TLV request			
Type	Length	Value	
0x29	0x02	gpio_idx	gpio_pull

Example:

TLV request			
Type	Length	Value	
		gpio_idx	gpio_pull, 0 -no pull 1 -pull down 3 -pull up
0x29	0x02	0x0D	0x01

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.16 dwm_gpio_value_set

4.3.16.1 Description

This API function sets the value of the GPIO pin to high or low.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
	gpio_value	8-bit integer	GPIO value, see Section 0 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.16.2 C code

Declaration:

```
int dwm_gpio_value_set(dwm_gpio_idx_t idx, bool value);
```

Example:

```
dwm_gpio_value_set(DWM_GPIO_IDX_13, 1);
```

4.3.16.3 SPI/UART Generic

Declaration:

TLV request			
Type	Length	Value	
0x2a	0x02	gpio_idx	gpio_value

Example:

TLV request			
Type	Length	Value	
		gpio_idx	gpio_value, 0 - low 1 - high
0x2a	0x02	0x0D	0x01

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.17 dwm_gpio_value_get

4.3.17.1 Description

This API function reads the value of the GPIO pin.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
Output	gpio_value	8-bit integer	GPIO value, see Section 0 for valid values.

4.3.17.2 C code

Declaration:

```
int dwm_gpio_value_get(dwm_gpio_idx_t idx, bool* p_value);
```

Example:

```
uint8_t value;
dwm_gpio_value_get(DWM_GPIO_IDX_13, &value);
printf("DWM_GPIO_IDX_13 value = %u\n", value);
```

4.3.17.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
0x2b	0x01	gpio_idx

Example:

TLV request		
Type	Length	Value
		gpio_idx
0x2b	0x01	0x0D

TLV response					
Type	Length	Value	Type	Length	Value
		err_code			gpio_value
0x40	0x01	0x00	0x55	0x01	0x01

4.3.18 dwm_gpio_value_toggle

4.3.18.1 Description

This API function toggles the value of the GPIO pin.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.18.2 C code

Declaration:

```
int dwm_gpio_value_toggle(dwm_gpio_idx_t idx);
```

Example:

```
dwm_gpio_value_toggle(DWM_GPIO_IDX_13);
```

4.3.18.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
0x2c	0x01	gpio_idx

Example:

TLV request		
Type	Length	Value
		gpio_idx
0x2c	0x01	0x0D

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.19 dwm_status_get

4.3.19.1 Description

This API function reads the system status: Location Data Ready. Location Data is the calculated result of the LE. When the tag finished TWR with the 4 anchors, the derived distance to the LE for position calculation. When this calculation is done, the Location Data Ready flag will be turn on. When the data is acquired by any API function, the flag will turn off until next calculation is done.

Parameter			Description
Field	Name	Size	
Input	none		
Output	status	8-bit integer	Bit0: loc_ready = '0' '1' (* new location data are ready *) Bit1: uwbmac_joined = '0' '1' (* node is connected to UWB network *)

4.3.19.2 C code

Declaration:

```
int dwm_status_get(dwm_status_t* p_status);
```

Example:

```
dwm_status_t status;
int rv;

rv = dwm_status_get(&status);
if (rv == DWM_OK) {
    printf("New location data: %d\n", status.loc_data);
    printf("UWB network joined: %d\n", status.uwbmac_joined);
} else {
    printf("error\n");
}
```

4.3.19.3 SPI/UART Generic

Declaration:

TLV request	
Type	Length
0x32	0x00

Example:

TLV request	
Type	Length
0x32	0x00

TLV response					
Type	Length	Value	Type	Length	Value
		err_code			loc_ready: yes uwbmac_joined: no
0x40	0x01	0x00	0x5A	0x01	0x01

4.3.20 dwm_int_cfg

4.3.20.1 Description

This API function enable interrupt generation for various events. Interrupts/events are communicated to the user by setting of dedicated GPIO pin (pin 19: READY), user can than read the status (dwm_status_get) and react according to the new status. The status is cleared when read. This call is available only on UART/SPI interfaces. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	Name	Size	
Input	none		
Output	data_ready_flag	8-bit integer	Bit 0: loc_ready. Interrupt is generated when location data are ready, 0=disable, 1=enable. Bit 1: spi_data_ready. Interrupt is generated when new SPI response data is available. 0=disable, 1=enable.

4.3.20.2 C code

This command is not available for on-board user application. It is used only available on external interfaces (UART/SPI).

4.3.20.3 SPI/UART Generic

Declaration:

TLV request		
Type	Length	Value
0x34	0x00	data_ready_flag

Example:

TLV request		
Type	Length	Value
		data_ready_flag, bit 0: loc_ready bit 1: spi_data_ready
0x34	0x01	0x03

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

4.3.21 dwm_gpio_irq_cfg

4.3.21.1 Description

This API function registers GPIO pin interrupt call back functions.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
	Irq_type	8-bit integer	interrupt type, 1 = rising, 2 = falling, 3 = both.
	p_cb	Function pointer	Pointer to the callback function on the interrupt.
	p_data	Data pointer	Pointer to data passed to the callback function.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.21.2 C code

Declaration:

```
int dwm_gpio_irq_cfg(dwm_gpio_idx_t idx, dwm_gpio_irq_type_t irq_type, dwm_gpio_cb_t* p_cb, void* p_data);
```

Example:

```
void gpio_cb(void* p_data)
{
    /* callback routine */
}

dwm_gpio_irq_cfg(DWM_GPIO_IDX_13, DWM_IRQ_TYPE_EDGE_RISING, &gpio_cb, NULL);
```

4.3.21.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

4.3.22 dwm_gpio_irq_dis

4.3.22.1 Description

This API function disables GPIO pin interrupt on the selected pin.

Parameter			Description
Field	Name	Size	
Input	gpio_idx	8-bit integer	GPIO number, see Section 3.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.22.2 C code

Declaration:

```
int dwm_gpio_irq_dis(dwm_gpio_idx_t idx);
```

Example:

```
dwm_gpio_irq_dis(DWM_GPIO_IDX_13);
```

4.3.22.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

4.3.23 dwm_i2c_read

4.3.23.1 Description

This API function read data from I2C slave.

Parameter			Description
Field	Name	Size	
Input	addr	8-bit integer	Address of a slave device (only 7 LSB)
	p_data	8-bit integer	Pointer to a receive data buffer
	len	8-bit integer	Number of bytes to be received
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.23.2 C code

Declaration:

```
int dwm_i2c_read(uint8_t addr, uint8_t* p_data, uint8_t len);
```

Example:

```
uint8_t data[2];
const uint8_t addr = 0x33; // some address of the slave device

dwm_i2c_read(addr, data, 2);
```

4.3.23.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

4.3.24 dwm_i2c_write

4.3.24.1 Description

This API function writes data to I2C slave.

Parameter			Description
Field	Name	Size	
Input	addr	8-bit integer	Address of a slave device (only 7 LSB)
	p_data	8-bit integer	Pointer to a receive data buffer
	len	8-bit integer	Number of bytes to be received
	no_stop	8-bit integer	0 or 1. If set to 1, the stop condition is not generated on the bus after the transfer has completed (allowing for a repeated start in the next transfer)
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

4.3.24.2 C code

Declaration:

```
int dwm_i2c_write(uint8_t addr, uint8_t* p_data, uint8_t len, bool no_stop);
```

Example:

```
uint8_t data[2];
const uint8_t addr = 1; // some address of the slave device

data[0] = 0xAA;
data[1] = 0xBB;
dwm_i2c_write(addr, data, 2, true);
```

4.3.24.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

4.3.25 dwm_evt_cb_register

4.3.25.1 Description

This API function registers event callback function. The callback function is called in case of any event that is intended for the user application. The user application decides whether to process the event and the data passed as input or if it ignores the event. This function applies only to on-board c code user application. Cannot be used with SPI or UART interface.

Parameter			Description
Field	Name	Size	
Input	p_cb	Function pointer	Pointer to the callback function on the event, input and output described below.
	p_data	Data pointer	Pointer to data passed to the callback function.
Output	err_code	8-bit integer	0, 1 or 2, see Section 3.4.1

Input and output data of event callback function. Input data is passed to the user in the callback function. The user can get data for actual event using event_id, see example in subsection 4.3.25.2.

Parameter			Description
Field	Name	Size	
Input	event_id	8-bit integer	Event ID to indicate the type of current event. 0 : new location data event, 1 : accelerometer event
	loc_data	location data	Location data available in new location data event. cnt, 8-bit integer: number of distances to the anchors. addr, 64-bit integer: UWB address/id of the particular anchor. distance, 32-bit integer: distance to particular anchor in millimetres. pqf, 8-bit integer: quality factor.
Output	none		

4.3.25.2 C code

Declaration:

```
void dwm_evt_cb_register(dwm_evt_cb_t* p_cb, void* p_data);
```

Example:

```
void on_dwm_evt(dwm_evt_t* p_evt, void* p_data)
{
    int i;

    switch (p_evt->header.id) {
    case DWM_EVT_NEW_LOC_DATA:
        printf("\nt:%lu ", dwm_system_us_get());
        /* process the data here */
        if (0 == p_evt->data.loc.p_pos) {
            /* location engine is turned off */
        } else {
            printf("[%ld,%ld,%ld,%u] ", p_evt->data.loc.p_pos->x,
                p_evt->data.loc.p_pos->y, p_evt->data.loc.p_pos->z,
                p_evt->data.loc.p_pos->qf);
        }

        for (i = 0; i < p_evt->data.loc.anchors.dist.cnt; ++i) {
            printf("%u", i);

            printf("0x%11x", p_evt->data.loc.anchors.dist.addr[i]);
            if (i < p_evt->data.loc.anchors.an_pos.cnt) {
                printf("[%ld,%ld,%ld,%u]",
                    p_evt->data.loc.anchors.an_pos.pos[i].x,
                    p_evt->data.loc.anchors.an_pos.pos[i].y,
                    p_evt->data.loc.anchors.an_pos.pos[i].z,
```

```
                p_evt->data.loc.anchors.an_pos.pos[i].qf);
            }

            printf("=%lu,%u ", p_evt->data.loc.anchors.dist.dist[i],
                p_evt->data.loc.anchors.dist.qf[i]);
        }
        printf("\n");
        break;

    default:
        break;
    }
}

...

int* p_user_data;
p_user_data = (int*)malloc(100);
dwm_evt_cb_register(on_dwm_evt, p_user_data);
...
/* if no user data are required */
dwm_evt_cb_register(on_dwm_evt, NULL);
```

4.3.25.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5 SHELL COMMANDS

5.1 Usage of UART Shell Mode

Shell mode shares UART interface with Generic mode. DWM1001 starts by default in UART Generic mode after reset. The Shell mode can be switched to by pressing ENTER twice within 1 second. The Generic mode can be switched to by executing command “quit” when in Shell mode. Shell mode and Generic mode can be switched back and forth.

Enter the Shell command and press “Enter” to execute the command. Press “Enter” without any command in Shell mode to repeat the last command. The following sub-sections provides overview of the Shell commands.

5.2 ?

Displays help.

Example:

```
dwm> ?
```

```
Usage: <command> [arg0] [arg1]
```

```
Build-in commands:
```

```
** Command group: Base **
```

```
?: this help
```

```
help: this help
```

```
quit: quit
```

```
** Command group: GPIO **
```

```
gc: GPIO clear
```

```
gg: GPIO get
```

```
gs: GPIO set
```

```
gt: GPIO toggle
```

```
** Command group: SYS **
```

```
f: Show free memory on the heap
```

```
ps: Show running threads
```

```
pms: Show PM tasks
```

```
reset: Reboot the system
```

```
si: System info
```

```
ut: Show device uptime
```

```
frst: Factory reset
```

```
** Command group: SENS **
```

```
twi: General purpose TWI read
```

```
aid: Read ACC device ID
```

```
av: Read ACC values
```

```
** Command group: LE **
```

```
les: Show meas. and pos.
```

```
lec: Show meas. and pos. in CSV
```

```
lep: Show pos. in CSV
```

**** Command group: UWBMAC ****

nmg: Get node mode
nmp: Set mode to PN (passive)
nmo: Set mode to PN (off)
nma: Set mode to AN
nmi: Set mode to AIN
nmt: Set mode to TN
nmtl: Set mode to TN-LP
bpc: Toggle BW/TxPWR comp
la: Show AN list
stg: Get stats
stc: Clear stats

**** Command group: API ****

tlv: Send TLV frame
aurs: Set upd rate
aurg: Get upd rate
apg: Get pos
aps: Set pos
acas: Set anchor config
acts: Set tag config

**** Tips ****

Press Enter to repeat the last command

5.3 help

Displays help, same as command "?".

Example:

```
dwm> help
... /*same output as command ? */
```

5.4 quit

Quit shell and switch UART into API mode.

Example:

```
dwm> quit
/* press enter twice to switch to shell mode again*/
```

5.5 gc

Clears GPIO pin. See section 3.4.3 for valid GPIO number.

Example:

```
dwm> gc
Usage: gc <pin>
dwm> gc 13
gpio13: 0
```

5.6 *gg*

Reads GPIO pin level. See section 3.4.3 for valid GPIO number.

Example:

```
dwm> gg
Usage: gg <pin>
dwm> gg 13
gpio13: 0
```

5.7 *gs*

Sets GPIO as output and sets its value. See section 3.4.3 for valid GPIO number.

Example:

```
dwm> gs
Usage: gs <pin>
dwm> gs 13
gpio13: 1
```

5.8 *gt*

Toggles GPIO pin (must be output). See section 3.4.3 for valid GPIO number.

Example:

```
dwm> gt
Usage: gt <pin>
dwm> gt 13
gpio13: 0
```

5.9 *f*

Show free memory on the heap.

Example:

```
dwm> f
[000014.560 INF] mem: free=3888 alloc=9184 tot=13072
```

5.10 *ps*

Show info about running threads.

Example:

```
dwm> ps
ID NAME          PRIO STACK USED
1 idle          31  512  304
2 ser           13  768  360
3 svc           15 1024  512
4 uwb           5 2560  760
5 api           10 2048 1180
6 sd            6 1024  352
7 ble           7 1024  392
8 le           14 2048  688
```

```
9 pm          17 512 304
10 sh         16 2048 1096
```

5.11 pms

Show power managements tasks. IDL means that task is idle. USE means that task is allocated in the power management.

Example:

```
dwm> pms
```

```
ID NAME      USE IDL ALM
0 ble        1  0  0
1 le         1  1  0
2 sh         1  0  0
3 --         0  0  0
4 --         0  0  0
```

5.12 reset

Reboot the system.

Example:

```
dwm> reset
/* node resets and boots in binary mode */
```

5.13 ut

Show device uptime.

Example:

```
dwm> ut
[000003.680 INF] uptime: 00:07:49.210 0 days (469210 ms)
```

5.14 frst

Factory reset.

Example:

```
dwm> frst
```

5.15 twi

General purpose I2C/TWI read.

Example: use twi to read accelerometer id. This should return the same value as using “aid” command. See section 5.16.

```
dwm> twi
```

Usage: twi <addr> <reg> [bytes to read (1 or 2)]

```
dwm> twi 0x33 0x0f 1
```

```
twi: addr=0x33, reg[0x0f]=0x33
```

5.16 *aid*

Read ACC device ID

Example:

```
dwm> aid
acc: 0x33
```

5.17 *av*

Read ACC values.

Example:

```
dwm> av
acc: x = 240, y = -3792, z = 16240
dwm> av
acc: x = 32, y = -3504, z = 15872
dwm> av
acc: x = 160, y = -3600, z = 16144
```

5.18 *les*

Show distances to ranging anchors and the position if location engine is enabled. Sending this command multiple times will turn on/off this functionality.

Example:

```
dwm> les
1151[5.00,8.00,2.25]=6.48 OCA8[0.00,8.00,2.25]=6.51 111C[5.00,0.00,2.25]=3.18
1150[0.00,0.00,2.25]=3.16 le_us=2576 est[2.57,1.98,1.68,100]
```

5.19 *lec*

Show distances to ranging anchors and the position if location engine is enabled in CSV format. Sending this command multiple times will turn on/off this functionality.

Example:

```
dwm> lec
DIST,4,AN0,1151,5.00,8.00,2.25,6.44,AN1,OCA8,0.00,8.00,2.25,6.50,AN2,111C,5.00,0.00,2.25,3.24,A
N3,1150,0.00,0.00,2.25,3.19,POS,2.55,2.01,1.71,98
```

5.20 *lep*

Show position in CSV format. Sending this command multiple times will turn on/off this functionality.

Example:

```
dwm> lep
POS,2.57,2.00,1.67,97
```


5.21 *si*

System info

Example:

```
dwm> si
[000077.340 INF] cfg:
[000077.340 INF] >fw1=x00022000
[000077.340 INF] board=DM1_A1
[000077.340 INF] cfg_ver=x00010500
[000077.350 INF] fw_ver=x01010200
[000077.350 INF] hw_ver=xDE410100
[000077.350 INF] opt=x03C10900
[000077.360 INF] fw_size[0]=x0001F000
[000077.360 INF] fw_size[1]=x0002F000
[000077.360 INF] fw_size[2]=x0002F000
[000077.370 INF] fw_csum[0]=x9445F89E
[000077.370 INF] fw_csum[1]=xF49DBF2A
[000077.370 INF] fw_csum[2]=xEB65F61F
[000077.380 INF] opt: LEDS LP UWB0 BLE I2C SPI UART
[000077.380 INF] mcu: temp=29.2 hfclk=rc:on lfclk=xtal:on
[000077.390 INF] uptime: 00:01:17.390 0 days (77390 ms)
[000077.390 INF] mem: free=2128 alloc=7224 tot=9352
[000077.400 INF] uwb: ch5 prf64 plen128 pac8 txcode9 rxcode9 sfd0 baud1
[000077.410 INF] uwb: tx_pwr=xC0/x25456585 125:250:500:norm[ms]=17:14:]
[000077.410 INF] uwb0: lna=0 xtaltrim=15 tx_delay=16472 rx_delay=16472
[000077.420 INF] uwb0: ID dev=xDECA0130 part=x1000113B lot=x10205EE9
[000077.430 INF] uwb0: panid=x0000 addr=xDECA7B1782D0113B
[000077.430 INF] mode: tn (act,twr,lp,nole)
[000077.440 INF] uwbmac: disconnected
[000077.440 INF] tn: upd_per=100000 upd_per_stat=100000 us
[000077.440 INF] tn: cnt=0 rtc:hrclk:devt dri=0.000000:0.000000:0.000000
```

5.22 *nmg*

Get node mode info.

Example:

```
dwm> nmg
mode: tn (act,twr,np,nole)
```

5.23 *nmo*

Enable passive offline option and resets the node.

Example:

```
dwm> nmo
```

```
/* press enter twice to switch to shell mode after reset*/  
DWM1001 TWR Real Time Location System
```

Copyright : 2016-2017 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg  
mode: tn (off,twr,np,nole)
```

5.24 nmp

Enable active offline option and resets the node.

Example:

```
dwm> nmp  
/* press enter twice to switch to shell mode after reset*/  
DWM1001 TWR Real Time Location System
```

Copyright : 2016-2017 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg  
mode: tn (pasv,twr,np,nole)
```

5.25 nma

Configures node to as anchor, active and reset the node.

Example:

```
dwm> nma  
/* press Enter twice */
```

DWM1001 TWR Real Time Location System

Copyright : 2016-2017 LEAPS and Decawave

License : Please visit https://decawave.com/dwm1001_license
Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg  
mode: an (act,-,-)
```

5.26 nmi

Configures node to as anchor - initiator, active and reset the node.

Example:

```
dwm> nmi  
/* press enter twice */
```

DWM1001 TWR Real Time Location System

Copyright : 2016-2017 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg  
mode: ain (act,real,-)
```

5.27 nmt

Configures node to as tag, active and reset the node.

Example:

```
dwm> nmt  
/* press enter twice */  
DWM1001 TWR Real Time Location System
```

Copyright : 2016-2017 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg
mode: tn (act,twr,np,le)
```

5.28 *nmtl*

Configures node to as tag, active, low power and resets the node.

Example:

```
dwm> nmtl
/* press enter twice */
DWM1001 TWR Real Time Location System
```

Copyright : 2016-2017 LEAPS and Decawave

License : Please visit https://decawave.com/dwm1001_license

Compiled : Jul 3 2017 12:33:40

Help : ? or help

```
dwm> nmg
mode: tn (act,twr,lp,le)
```

5.29 *bpc*

Toggle UWB bandwidth / tx power compensation.

Example:

```
dwm> bpc
BW/TxPwr comp off
dwm> bpc
BW/TxPwr comp on
```

5.30 *la*

Show anchor list.

Example:

```
dwm> la
[003452.590 INF] AN: cnt=4 seq=x03
[003452.590 INF] 0) id=DECA5419E2E01151 seat=0 idl=0 seens=0 col=0 cl=000F nbr=000F fl=4002
opt=03C1B046 hw=DE41010
0 fw=01010101 fwc=7B033F01
[003452.600 INF] 1) id=DECA78A55EB00CA8 seat=1 idl=1 seens=119 col=0 cl=000F nbr=0000
fl=0000 opt=03C1B042 hw=DE410
```

```
100 fw=01010101 fwc=7B033F01
[003452.620 INF] 2) id=DECAFCED8D50111C seat=2 idl=1 seens=103 col=0 cl=000F nbr=0000
fl=0000 opt=03C1B042 hw=DE410
100 fw=01010101 fwc=7B033F01
[003452.640 INF] 3) id=DECA51421A901150 seat=3 idl=0 seens=81 col=0 cl=000F nbr=0000 fl=0000
opt=03C1B042 hw=DE4101
00 fw=01010101 fwc=7B033F01
```

5.31 *stg*

Displays statistics.

Example:

```
dwm> stg
uptime: 0
memfree: 0
mcu_temp: 0
rtc_drift: 0.000000
ble_con_ok: 0
ble_dis_ok: 0
ble_err: 0
uwb0_intr: 0
uwb0_rst: 0
rx_ok: 56454
rx_err: 0
tx_err: 0
tx_errx: 0
bcn_tx_ok: 1661
bcn_tx_err: 0
bcn_rx_ok: 0
alma_tx_ok: 51
alma_tx_err: 0
alma_rx_ok: 0
cl_tx_ok: 0
cl_tx_err: 0
cl_rx_ok: 0
cl_coll: 0
fwup_tx_ok: 0
fwup_tx_err: 0
fwup_rx_ok: 0
svc_tx_ok: 0
svc_tx_err: 0
svc_rx_ok: 0
clk_sync: 0
twr_ok: 0
twr_err: 0
```

```
res[0]: 0
res[1]: 0
res[2]: 0
res[3]: 0
res[4]: 0
res[5]: 0
res[6]: 0
res[7]: 0
tot_err: 0
```

5.32 stc

Clears statistics.

Example:

```
dwm> stc
```

5.33 tlv

Parses given TLV frame. See Section 4 for valid TLV commands.

Example: read node configuration

```
dwm> tlv 8 0
```

OUTPUT FRAME:

```
40 01 00 46 02 b0 00
```

Example: toggles GPIO pin 13

```
dwm> tlv 44 1 13
```

OUTPUT FRAME:

```
40 01 00
```

5.34 aurs

Set position update rate. See section 4.3.3 for more detail.

Example:

```
dwm> aurs
```

Usage aurs <ur> <urs>

```
dwm> aurs 10 20
```

```
err code: 0
```

5.35 aurg

Get position update rate. See section 4.3.4 for more detail.

Example:

```
dwm> aurg
```

```
err code: 0, upd rate: 10, 20(stat)
```

5.36 *apg*

Get position of the node. See section 3.4.2 for more detail.

Example:

```
dwm> apg
x:100 y:120 z:2500 qf:100
```

5.37 *aps*

Set position of the node. See section 3.4.2 for more detail.

Example:

```
dwm> aps
Usage aps <x> <y> <z>
dwm> aps 100 120 2500
err code: 0
```

5.38 *acas*

Configures node as anchor with given options. See section 3.4.8 for more detail. Note that this command only sets the configuration parameters. To make effect of the settings, users should issue `s` reset command, see section 5.12 for more detail.

Example:

```
dwm> acas
Usage acas <inr> <bridge> <leds> <ble> <uwb> <fw_upd>
dwm> acas 0 0 1 1 2 0
err code: 0
```

5.39 *acts*

Configures node as tag with given options. See section 3.4.7 for more detail. Note that this command only sets the configuration parameters. To make effect of the settings, users should issue `s` reset command, see section 5.12 for more detail.

Example:

```
dwm> acts
Usage acts <meas_mode> <accel_en> <low_pwr> <loc_en> <leds> <ble> <uwb> <fw_upd>
dwm> acts 0 1 0 1 1 0 0 0
err code: 0
```

6 APPENDIX A – TLV TYPE LIST

Table 4 DWM1001 TLV type list

Type		Comment
Name	Value	
TYPE_CMD_POS_SET	1	Request: set position coordinates XYZ
TYPE_CMD_POS_GET	2	Request: get position coordinates XYZ
TYPE_CMD_UR_SET	3	Request: set position update rate
TYPE_CMD_UR_GET	4	Request: get position update rate
TYPE_CMD_CFG_TN_SET	5	Request: set configuration for the tag
TYPE_CMD_CFG_AN_SET	7	Request: set configuration for the anchor
TYPE_CMD_CFG_GET	8	Request: get configuration data
TYPE_CMD_CFG_SAVE	9	Request: save configuration data
TYPE_CMD_SLEEP	10	Request: sleep
TYPE_CMD_LOC_GET	12	Request: location get
TYPE_CMD_BLE_ADDR_SET	15	Request: BLE address set
TYPE_CMD_BLE_ADDR_GET	16	Request: BLE address get
TYPE_CMD_RESET	20	Request: reset
TYPE_CMD_VER_GET	21	Request: get FW version
TYPE_CMD_GPIO_CFG_OUTPUT	40	Request: configure output pin and set
TYPE_CMD_GPIO_CFG_INPUT	41	Request: configure input pin
TYPE_CMD_GPIO_VAL_SET	42	Request: set pin value
TYPE_CMD_GPIO_VAL_GET	43	Request: get pin value
TYPE_CMD_GPIO_VAL_TOGGLE	44	Request: toggle pin value
TYPE_CMD_STATUS_GET	50	Request: get status
TYPE_CMD_INT_CFG	52	Request: configure interrupts
TYPE_RET_VAL	64	Response: the previous request is ok or error
TYPE_POS_XYZ	65	Response: position coordinates x,y,z
TYPE_POS_X	66	Response: position coordinate x
TYPE_POS_Y	67	Response: position coordinate y
TYPE_POS_Z	68	Response: position coordinate z
TYPE_UR	69	Response: update rate
TYPE_CFG	70	Response: configuration data
TYPE_DIST	72	Response: distances
TYPE_RNG_AN_POS_DIST	73	Response: ranging anchor distances and positions
TYPE_FW_VER	80	Response: firmware version
TYPE_CFG_VER	81	Response: configuration version
TYPE_HW_VER	82	Response: hardware version
TYPE_PIN_VAL	85	Response: pin value
TYPE_STATUS	90	Response: status
TYPE_CMD_N_POS_SET	128	Request: nested command set position
TYPE_CMD_N_LOC_GET	130	Request: nested command location get
TYPE_DUMMY	0	Reserved for SPI dummy byte

7 APPENDIX B – BIBLIOGRAPHY

1. nRF5 Getting Started, available from www.nordicsemi.com
2. 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/>
3. DWM1001 Firmware User Guide, available from www.decawave.com
4. DWM1001 System Overview, available from www.decawave.com
5. DWM1001 on-board package, available from www.decawave.com
6. DWM1001 Host API package, available from www.decawave.com

8 DOCUMENT HISTORY

Table 5: Document History

Revision	Date	Description
1.0	18-Dec-2017	Initial release.

9 ABOUT DECAWAVE

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

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

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

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

<mailto:sales@decawave.com>

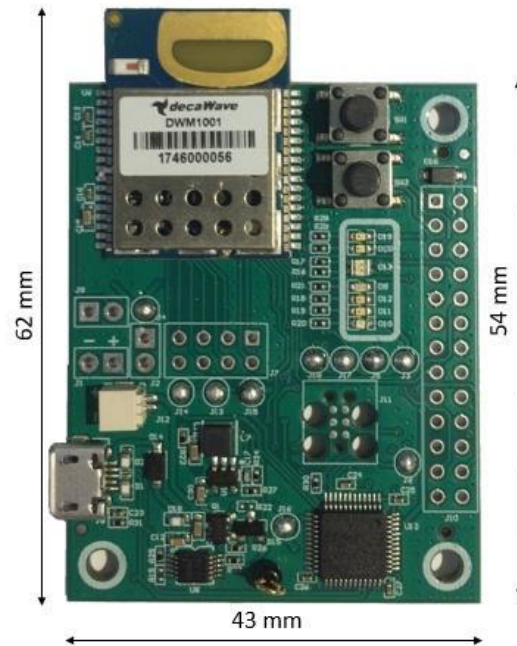
<http://www.decawave.com/>



Product Overview: DWM1001-DEV

DWM1001 Module Development Board

- Plug-and-Play Development Board for evaluating the performance of the Decawave DWM1001 module
- Easily assemble a fully wireless RTLS system, including anchors, tags & gateways, without designing any hardware or writing a single line of code – and quickly progress into developing your application



Key Features and Benefits

- DWM1001 module mounted (See DWM1001 data sheet for details)
- USB connection for reprogramming, debug & power supply
- On board JLINK
- External API via SPI, UART & BLE for configuration & control
- 26-pin Raspberry Pi compatible header
- Reset and user-defined buttons and LEDs
- Battery charging circuit
- Allows access to DWM1001 pins (castellation) via on board headers

Table of Contents

1 OVERVIEW	4	7 BUTTONS SW1 AND SW2	11
1.1 SUITABLE POWER SUPPLY OPTIONS	5	8 DEVELOPMENT BOARD SOLDER BRIDGE	
1.2 BLOCK DIAGRAM OF THE DEVELOPMENT BOARD .	5	JUMPERS	12
2 DWM1001 MODULE.....	6	9 DEVELOPMENT BOARD SCHEMATIC	13
3 RASPBERRY PI INTERFACE	6	10 REFERENCES	14
3.1 MEANS OF CONNECTION.....	6	11 DOCUMENT HISTORY	14
3.2 DWM1001 MODULE PIN TO RASPBERRY PI		12 MAJOR CHANGES	14
CONNECTOR MAPPING	7	13 ABOUT DECAWAVE	15
4 THE DEVELOPMENT BOARD LEDS.....	9		
5 ON BOARD JLINK.....	10		
6 POWER SUPPLY AND BATTERY CONNECTIONS	11		

List of Figures

FIGURE 1 THE MAIN COMPONENTS OF THE MODULE DEVELOPMENT BOARD ARE SHOWN	4	FIGURE 7: HEADER CONNECTOR WITH EXTENDED PIN LENGTHS	7
FIGURE 2: THE MODULE DEVELOPMENT BOARD CAN BE USED TO CREATE AN ANCHOR, A TAG OR A GATEWAY	4	FIGURE 8: PIN DESIGNATIONS OF THE RASPBERRY PI MODEL A VARIANT.....	8
FIGURE 3: MODULE DEVELOPMENT BOARD USED AS AN ANCHOR, TAG OR GATEWAY DEVICE, IN AN RTLS SYSTEM.....	4	FIGURE 9: PIN DESIGNATIONS OF THE RASPBERRY PI MODEL B VARIANT.....	8
FIGURE 4: THE MAIN SECTIONS OF THE DWM1001 MODULE DEVELOPMENT BOARD	5	FIGURE 10: FRONT VIEW OF THE DWM1001-DEV MODULE DEVELOPMENT BOARD	9
FIGURE 5: BLOCK DIAGRAM OF DWM1001 MODULE	6	FIGURE 11: MODULE DEVELOPMENT BOARD JLINK COMPONENTS	10
FIGURE 6: RASPBERRY PI MODEL A TYPE WITH RIBBON CABLE FOR CONNECTION TO THE MODULE DEVELOPMENT BOARD	6	FIGURE 12: PICTURE OF THE MODULE DEVELOPMENT BOARD SHOWING THE BATTERY CONNECTION POINTS AND BUTTONS.....	11

List of Tables

TABLE 1: POSSIBLE SOURCES OF POWER FOR THE MODULE DEVELOPMENT BOARD	5
TABLE 2: CONNECTIONS BETWEEN RASPBERRY PI AND DWM1001 MODULE	7
TABLE 3: THE FIRMWARE INDICATION LEDS	9
TABLE 4: A LIST OF SOLDER JUMPERS AVAILABLE ON THE MODULE DEVELOPMENT BOARD	12
TABLE 5: DOCUMENT HISTORY.....	14

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 with Decawave for the most recent updates on this product.

The DWM1001 module mounted on the DWM1001-DEV PCB is pre-loaded with firmware, please refer to the "DWM1001 Firmware User Guide" for disclaimer and license terms.

Copyright © 2017 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.

REGULATORY APPROVALS

The DWM1001, as supplied from Decawave, has not been certified for use in any particular geographic region by the appropriate regulatory body governing radio emissions in that region although it is capable of such certification depending on the region and the manner in which it is used.

All products developed by the user incorporating the DWM1001 must be approved by the relevant authority governing radio emissions in any given jurisdiction prior to the marketing or sale of such products in that jurisdiction and user bears all responsibility for obtaining such approval as needed from the appropriate authorities.

1 OVERVIEW

This document gives technical details of the DWM1001 module development board, called the DWM1001-DEV. All the functions of the DWM1001 module can be exercised with this board. Figure 1 gives an overview of the main components of the module development board.

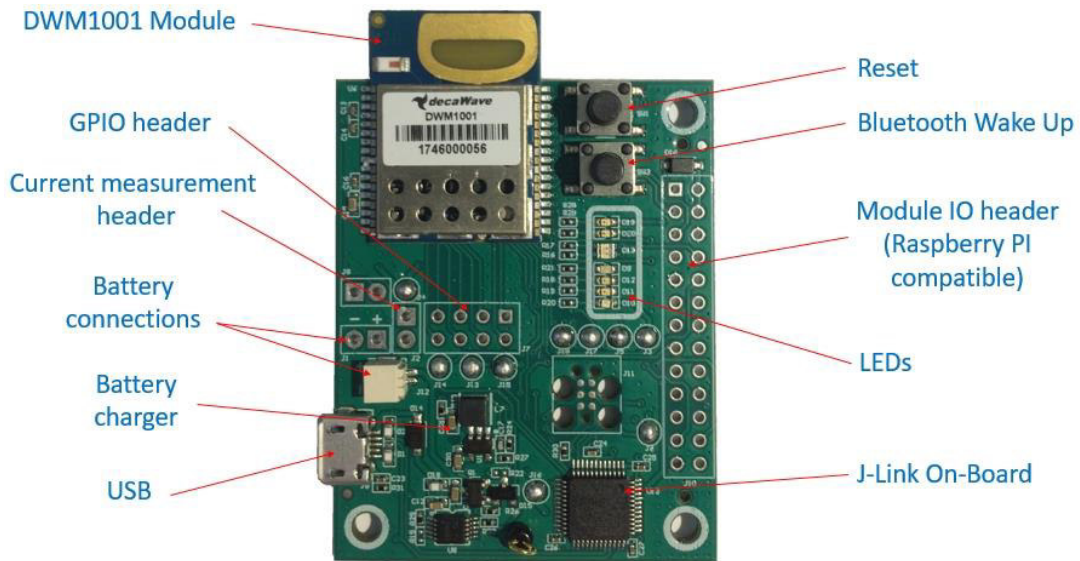
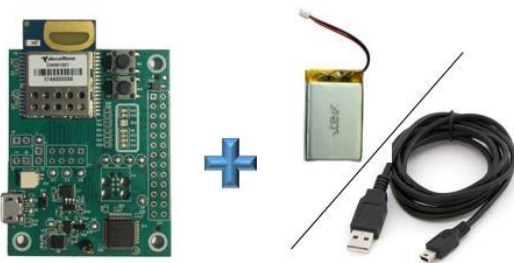


Figure 1 The main components of the module development board are shown

The module development board can be used to create an Anchor or a Tag for an RTLS system. This is shown in Figure 2. It can also be combined with a Raspberry Pi to create a gateway device. Figure 3 shows the configuration of an RTLS system where the module development board can be an Anchor, Tag or Gateway device.

Build an Anchor or a Tag



Build a Gateway

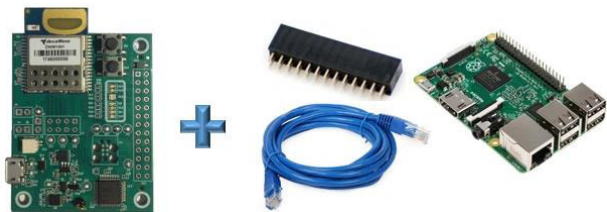


Figure 2: The module development board can be used to create an Anchor, a Tag or a Gateway

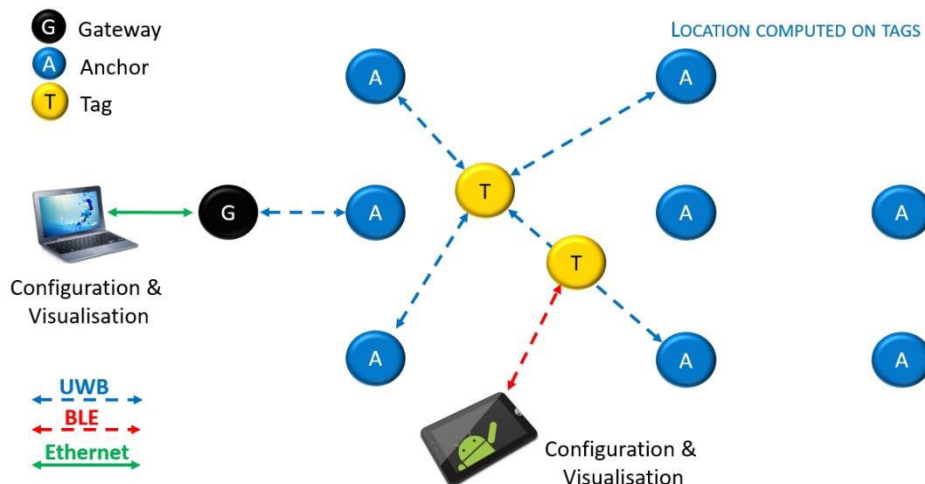


Figure 3: Module development board used as an Anchor, Tag or Gateway device, in an RTLS system

1.1 Suitable Power Supply Options

The module development board has a voltage supply requirement of 3.6V to 5.5V. The module development board can be powered from three different sources. Details are given in Table 1.

Table 1: Possible sources of power for the module development board

Power Source	Voltage level	Current level (Recommended)	Notes
USB Connection	+5V	500mA	The board requires a connection to a high power USB connection. Check that it can supply at least 500mA.
Battery	3.6V - 5.5V	500mA	Any battery that meets the 3.6V to 5.5V voltage supply will suffice.
Raspberry Pi Power	+5V	500mA	

1.2 Block Diagram of the development board

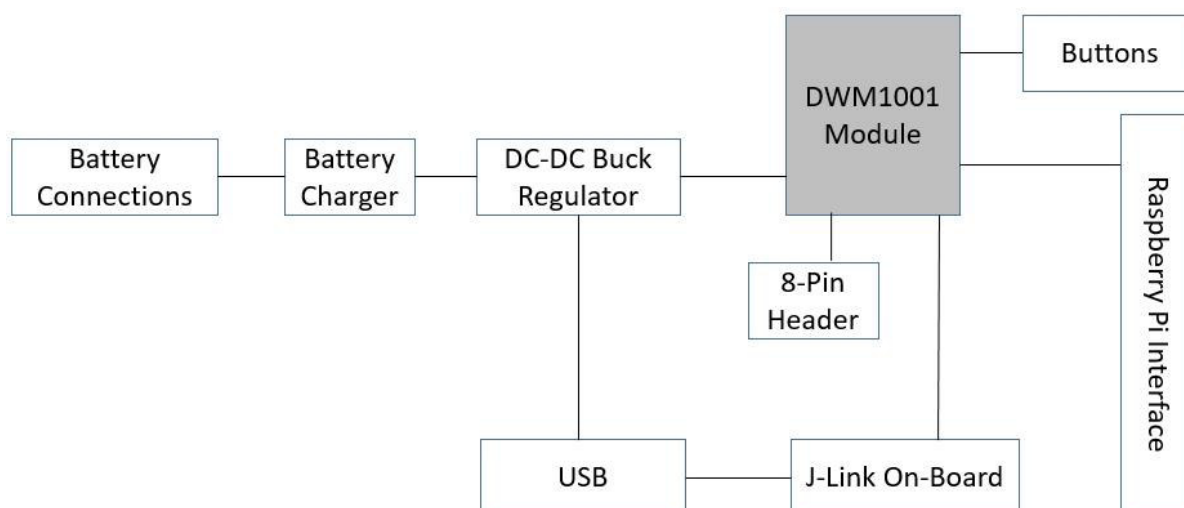


Figure 4: The main sections of the DWM1001 Module Development Board

Figure 4 shows the main sections of the Module Development Board. A brief overview of these sections is given below with further details given in later sections of this document.

The DWM1001 module is based on Decawave's DW1000 Ultra Wideband (UWB) transceiver IC, which is an IEEE 802.15.4-2011 UWB implementation. It integrates UWB and Bluetooth antenna, all RF circuitry, Nordic Semiconductor nRF52832 and motion sensor [1].

The USB connection can provide power to the Module Development Board and also allows for the capability to flash the DWM1001 module and furthermore to debug software running on the DWM1001 module.

The Power Supply takes its input from USB or from a Battery or from a connected Raspberry Pi. It powers the module and the other devices on the Module Development Board. It can also charge a connected battery when powered by USB or the Raspberry Pi.

Two buttons and a number of LEDs are provided for end user applications. A header to interface to the Raspberry Pi is also provided.

2 DWM1001 MODULE

Figure 5 shows a block diagram of the module. All major sections of the module are shown, along with the source of signals coming to the module's pins.

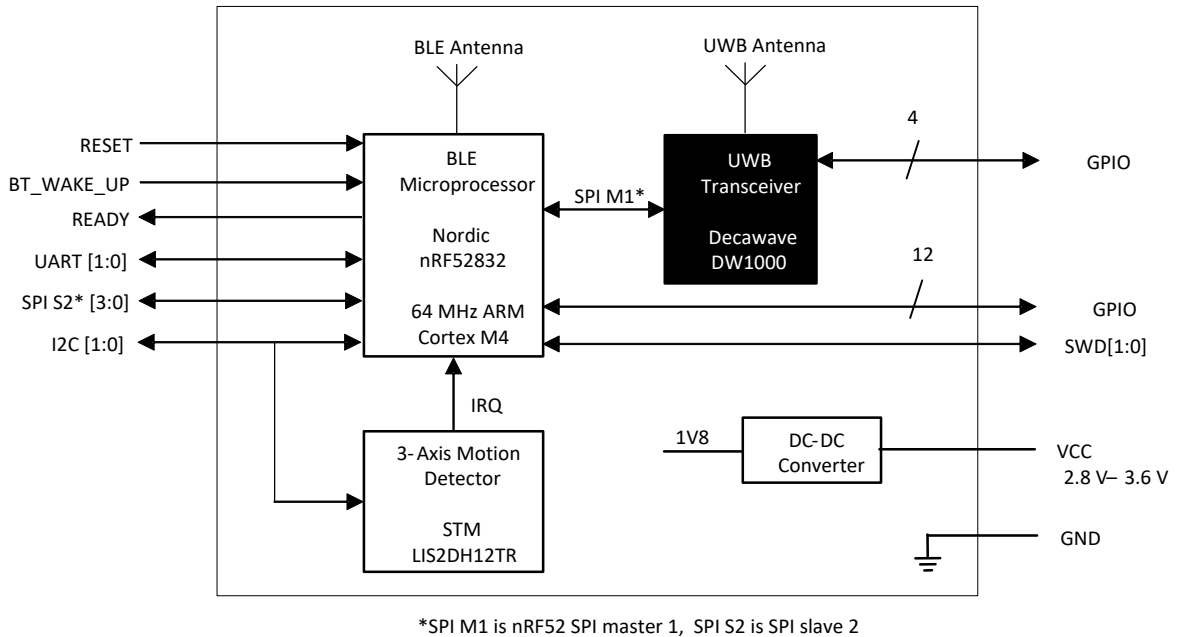


Figure 5: Block diagram of DWM1001 module

3 RASPBERRY PI INTERFACE

3.1 Means of connection

There are a number of types of Raspberry Pi. The preferred options are the A and B variants. To use the A variant you will require a ribbon cable to connect to the Module Development Board. Figure 6 below shows a Raspberry Pi Model A with the ribbon cable connected.

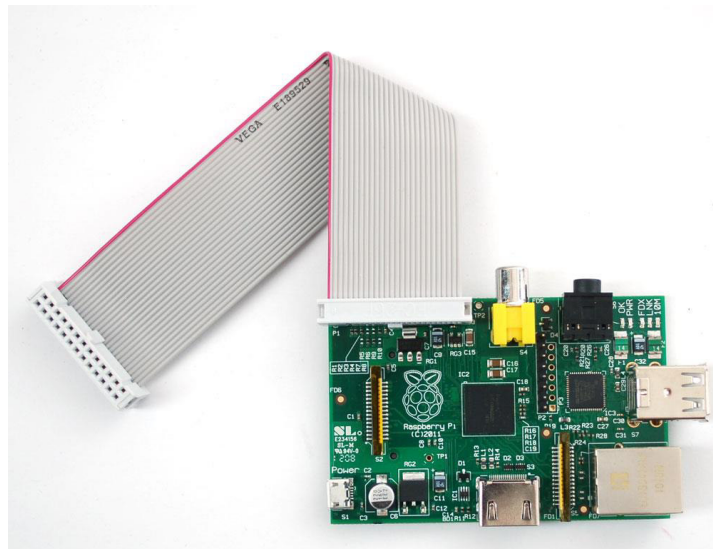


Figure 6: Raspberry Pi Model A Type with ribbon cable for connection to the Module Development Board

As an alternative to the ribbon cable it is possible to get header connectors with extra long pins. Such a connector is shown in Figure 7. One supplier of these connectors is <https://www.modmypi.com>. Search for part number MMP-0275.

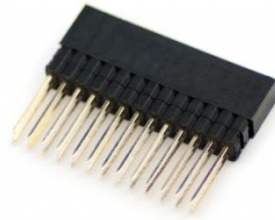


Figure 7: Header connector with extended pin lengths

DWM1001 Module Pin to Raspberry Pi Connector Mapping Table 2 below gives the connection details between the Raspberry Pi connector and the DWM1001 module. See the DWM1001 datasheet[1] and the schematic for the DWM1001 Module Development Board at the end of this document.

Table 2: Connections between Raspberry Pi and DWM1001 module

Module Development Board RPi connector		Module Pin Number (and Name) from DWM1001 Module Data Sheet
Pin Number	Schematic Net Name	
3	SDA_RPI	Pin 23 (GPIO_15)
5	SCL_RPI	Pin 25 (GPIO_8)
9	GND	GND
15	GPIO_RPI	Pin 19 (READY)
19	SPI1_MOSI	Pin 27 (SPIS_MOSI)
21	SPI1_MISO	Pin 26 (SPIS_MISO)
23	SPI1_CLK	Pin 25 (GPIO_8)
25	GND	GND
2	VRPI	Provides input power to Module Development Board. (Not connected directly to module)
4	VRPI	
6	GND	
8	TXD	Pin 18 (UART_RX)
10	RXD_RPI/RXD	Pin 20 (UART_TX)
12	RESET	Pin 33 (RESETn)
14	GND	GND
20	GND	GND
24	CS_RPI	Pin29 (SPIS_CSn)

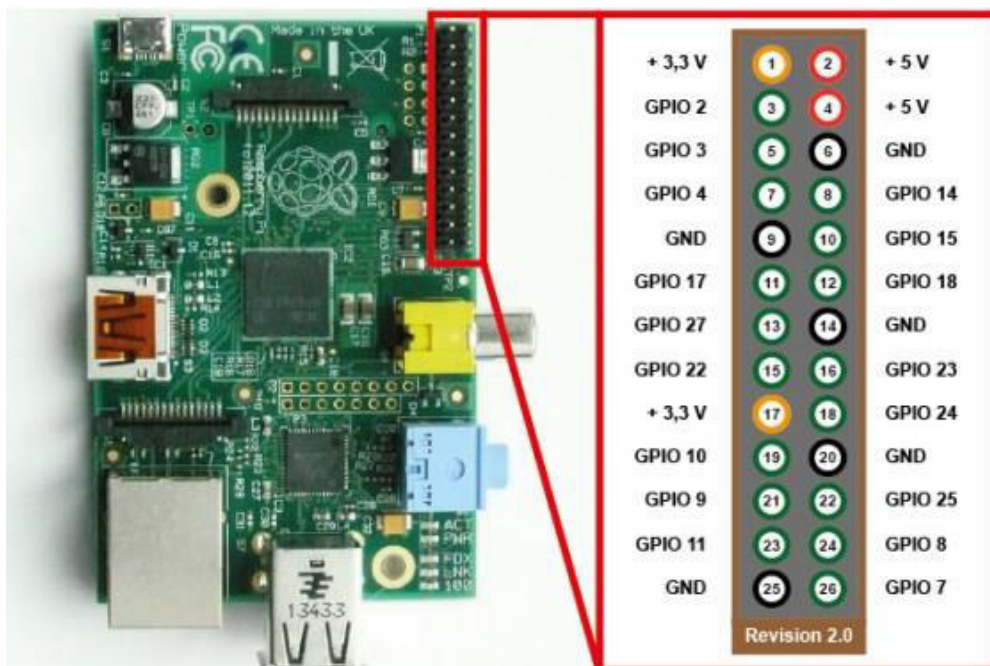


Figure 8: Pin designations of the Raspberry Pi Model A variant.

Figure 8 shows the pin designations for the model A Raspberry Pi variant. The Raspberry Pi Model B variant has a larger header connector with 40 pins. Figure 9 below shows its header and connector pin designations.

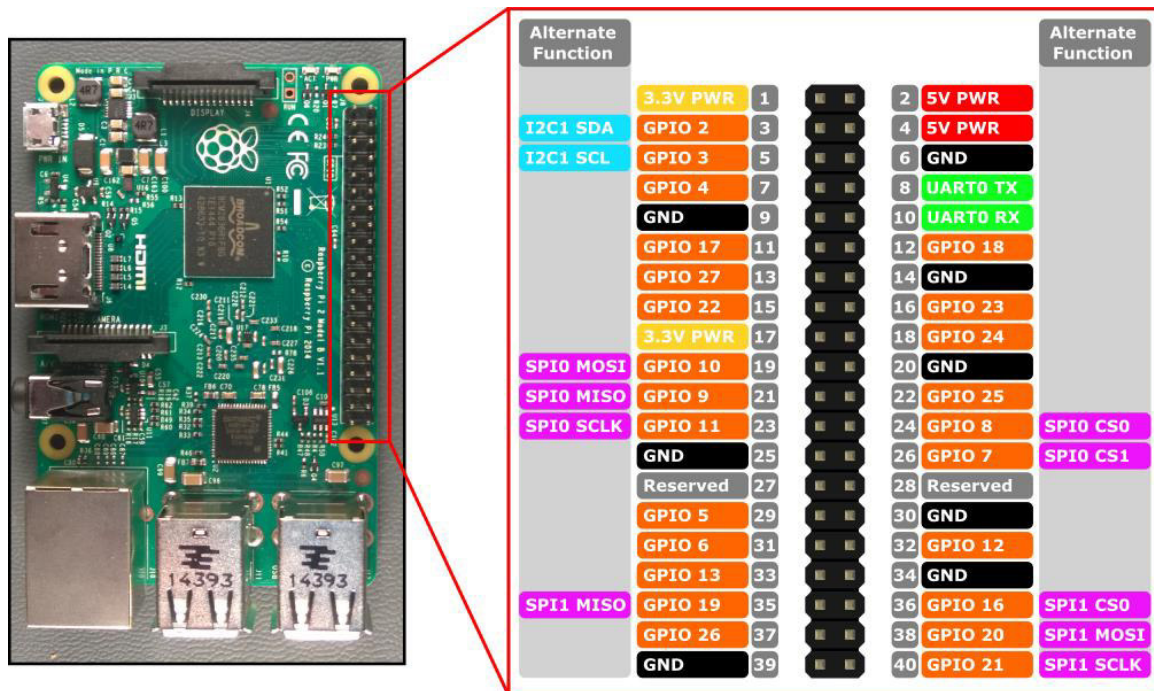


Figure 9: Pin designations of the Raspberry Pi Model B variant.

4 THE DEVELOPMENT BOARD LEDs

The Development board has a number of LEDs for indication purposes. They give useful indication of a number of events within the system. Figure 10 shows the LEDs and their use with the pre programmed firmware. Table 3 gives further specific details on D9, D10 and D11.



Figure 10: Front View of the DWM1001-DEV Module Development Board

Note (1): See DW1000 IC User Manual for description of D13 TX/RX LEDs

Note (2): F/W function of LEDs D9, D11, D10 is shown in the table below

Table 3: The Firmware Indication LEDs

GREEN D9	RED D11	BLUE D10	Function	Description
Blink Fast	Blink Fast	Blink Fast	Bootloader Active	LEDs blink twice
Blink Fast	Blink Fast	Blink Fast	Firmware Update in Progress	LEDs alternate between Blue and Green/Red
On	On		Mode Status	UWB Passive Mode
On	On			UWB Off Mode
Blink Fast	Blink Fast			Anchor Node (no UWB signal detected for more than 8 s)
Blink Slow	Blink Slow			Tag Node (no UWB signal detected for more than 8 s)
On				Tag Low Power Mode: ON
Off				Tag Low Power Mode: SLEEP
On			MAC Status	Connected anchor or tag
Blink Slow				Connected anchor initiator
Blink Fast				UWB communication detected
Off				Low Power node: no signal detected for more than 6 s
-		On	Bluetooth Status	Bluetooth Connected
-		Off		Bluetooth Disconnected
-	On		Data / Measurement Status	UWB TX/RX Active
-	Off			Idle

5 ON BOARD JLINK

The processor on the Module Development Board provides USB to SWD (Serial Wire Debug) conversion to allow programming and debug of software on the DWM1001 module. Figure 11 below shows the relevant sections on the Module Development Board.

Serial Wire Debug is a replacement for the more traditional 5-pin JTAG port. It uses a clock (SWDCLK) and a Single bi-directional data pin (SWDIO), providing all the normal JTAG debug and test functionality. SWDIO and SWCLK are overlaid on the TMS and TCK pins. In order to communicate with a SWD device, J-Link sends out data on SWDIO, synchronous to the SWCLK. With every rising edge of SWCLK, one bit of data is transmitted or received on the SWDIO. The data read from SWDIO can then be retrieved from the input buffer.

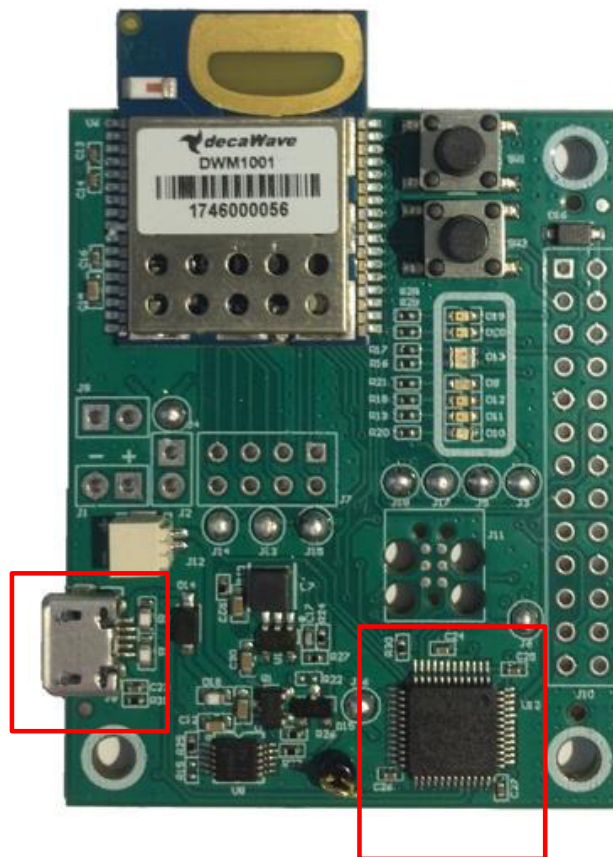


Figure 11: Module Development Board JLink components

6 POWER SUPPLY AND BATTERY CONNECTIONS

The Power Supply takes its input from USB or from a Battery or from a connected Raspberry Pi. It powers the module and the other devices on the Module Development Board. It can also charge a connected battery when powered by USB or the Raspberry Pi.

The Battery Charger is a Lithium - Ion battery charger. Batteries can be connected to the module development board at the connectors shown in Figure 12.

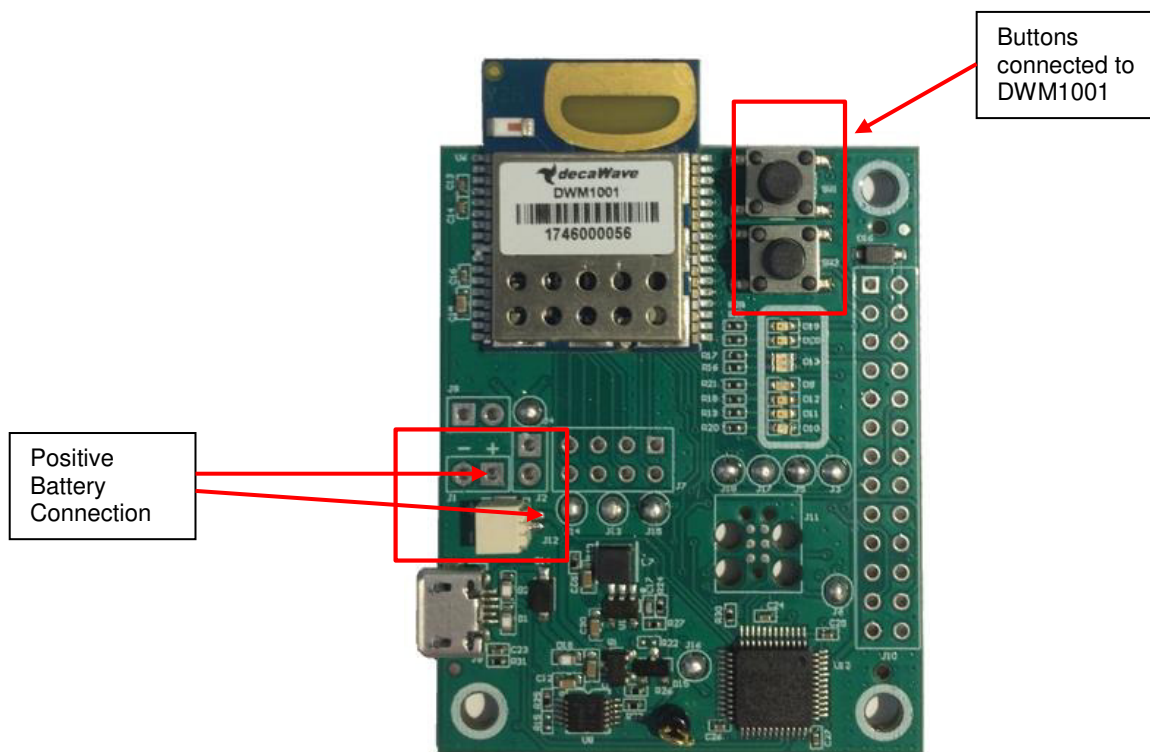


Figure 12: Picture of the module development board showing the battery connection points and buttons

7 BUTTONS SW1 AND SW2

There are two buttons on the PCB, SW1 and SW2. SW1 is connected to the RESETn pin on the DWM1001 module and SW2 is connected to the BT_WAKE_UP pin on the DWM1001 module. SW2 wakes up the Bluetooth functionality when a tag is in low-power mode, as described in the System Overview document[3]. Buttons are shown above in Figure 12.

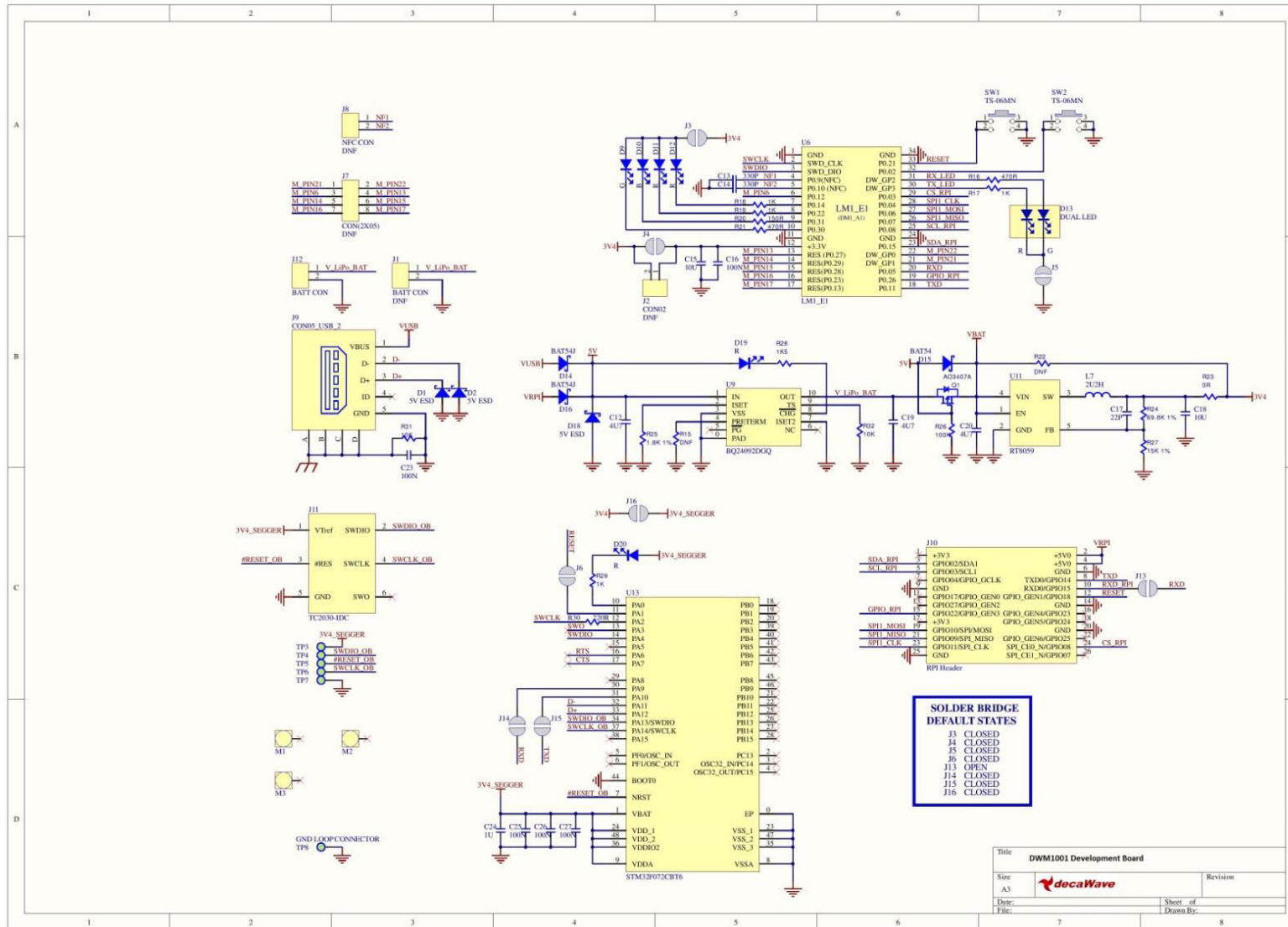
8 DEVELOPMENT BOARD SOLDER BRIDGE JUMPERS

The Module Development Board has eight solder bridge jumpers. These can be used to allow evaluation of different aspects of the DWM1001 modules performance. For example Solder Bridge J4 can be desoldered and a resistor placed across connector J2 to allow measurement of the modules current consumption. Table 4 gives a list of these jumpers, their purpose and the state they are in when leaving the factory. Investigation of the schematic of the Module Development board at the end of this document will give further details on their use.

Table 4: A list of solder jumpers available on the Module Development Board

Jumper Number	Purpose	Default State
J3	Desolder to disconnect user LEDs from the module	Closed
J5	Desolder to disconnect Tx and Rx LEDs from the module	Closed
J4	Desolder to measure module current in J2	Closed
J6	Desolder to disconnect Reset button from JLINK	Closed
J13	Solder to connect UART Rx between Module and Raspberry Pi	Closed
J14	Desolder to disconnect module RXD from JLINK	Closed
J15	Desolder to disconnect module TXD from JLINK	Closed
J16	Desolder to disconnect power to JLINK	Closed

9 DEVELOPMENT BOARD SCHEMATIC



10 REFERENCES

- [1] Decawave DWM1001 Datasheet www.decawave.com
- [2] IEEE802.15.4-2011 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/>
- [3] DWM1001 System Overview

11 DOCUMENT HISTORY

Table 5: Document History

Revision	Date	Description
1.0	20/12/17	First Version

12 MAJOR CHANGES

To be completed when document is updated.

13 ABOUT DECAWAVE

Decawave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4-2011[2] UWB standard. This device is the first in a family of parts that will operate at data rates of 110 kbps, 850 kbps, 6.8 Mbps.

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

Further Information

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

Decawave Ltd
Adelaide Chambers
Peter Street
Dublin
D08 T6YA
Ireland
+353 1 6975030
e: sales@decawave.com
w: www.decawave.com

APS013 APPLICATION NOTE

The implementation of two-way ranging with the DW1000

Version 2.2

This document is subject to change without notice

TABLE OF CONTENTS

LIST OF TABLES2

LIST OF FIGURES.....2

1 INTRODUCTION.....3

1.1 DW1000 BASED TWR 3

2 IMPLEMENTATION OF RANGING.....5

2.1 DISCOVERY PHASE..... 5

2.2 RANGING PHASE 5

2.3 MESSAGES USED IN RANGING..... 6

2.3.1 *General ranging frame format*..... 6

2.3.2 *Blink frame format*..... 8

2.3.3 *Poll message* 8

2.3.4 *Response message*..... 8

2.3.5 *Final message* 8

2.3.6 *Ranging Initiation message* 9

2.4 TWR OPTIMISATION FOR POWER CONSUMPTION 9

2.4.1 *Discovery phase* 9

2.4.2 *Ranging phase* 11

3 CONCLUSION.....13

4 REFERENCES13

5 DOCUMENT HISTORY13

6 MAJOR CHANGES13

7 ABOUT DECAWAVE15

LIST OF TABLES

TABLE 1: FIELDS WITHIN THE RESPONSE MESSAGE..... 8

TABLE 2: FIELDS WITHIN THE FINAL MESSAGE..... 9

TABLE 3: FIELDS WITHIN THE RANGING INITIATION MESSAGE 9

TABLE 4: TABLE OF REFERENCES 13

TABLE 5: DOCUMENT HISTORY..... 13

LIST OF FIGURES

FIGURE 1: TWO-WAY RANGING CONCEPT 3

FIGURE 2: ASYMMETRIC TWR TOF FORMULA 4

FIGURE 3: DISCOVERY AND RANGING PHASE MESSAGE EXCHANGES 5

FIGURE 4: GENERAL RANGING FRAME FORMAT..... 7

FIGURE 5: RANGING MESSAGE ENCODINGS 8

FIGURE 6: TWR DISCOVERY PHASE TIMING PROFILE 10

FIGURE 7: TWR RANGING PHASE TIMING PROFILE..... 12

1 INTRODUCTION

In this application note two-way ranging (TWR) scheme as used by Decawave’s example application (*DecaRanging*) is described. TWR is a basic concept to calculate the distance between two objects by determining the time of flight (TOF) of signals travelling between them.

The distance between the objects may be calculated using the formula,

$$Distance = Speed\ of\ radio\ waves \times TOF$$

The DW1000 uses mathematical and electronic techniques to implement a very precise clock. By recording the state of this clock when certain events occur during DW1000 transmission and reception of the radio wave signals, the DW1000 has the ability to ‘timestamp’ those events.

TWR has advantages over other distance measurement and locating systems in that it can be used by stand-alone devices which only have relative distances to measure. There is no requirement for an infrastructure of fixed communicating devices to determine separation distances.

1.1 DW1000 based TWR

If we use a pair of DW1000s, designated as an initiator and a responder respectively, we can describe the two-way ranging concept as follows.

The initiator transmits a radio message to the responder and records its time of transmission (transmit timestamp) t_1 . The responder receives the message and transmits a response (a radio message) back to the initiator after a particular delay t_{reply} . The initiator then receives this response and records a receive timestamp t_2 . This process is depicted in Figure 1.

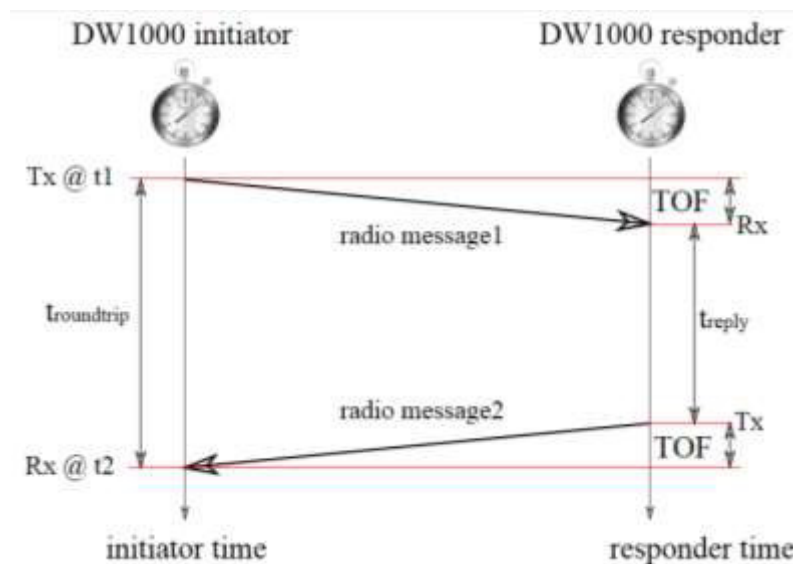


Figure 1: Two-way ranging concept

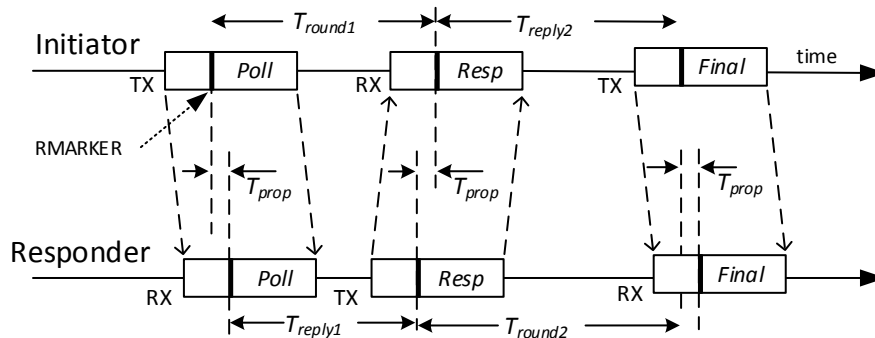
Now using the timestamps t_1 and t_2 , the initiator can calculate the round trip time $t_{roundtrip}$ and knowing the reply time in the tag, t_{reply} , the TOF can be determined by,

$$TOF = \frac{t_2 - t_1 - t_{reply}}{2}$$

If we assume the speed of radio waves through air is the same as the speed of light c , then the distance between the initiator and responder can be calculated by,

$$Distance = c \times \frac{t_2 - t_1 - t_{reply}}{2}$$

In the case of tag-to-anchor two-way ranging, there are a number of sources of error due to clock drift and frequency drift [4]. Asymmetric double sided TWR method is used in Decawave's implementation. It reduces the error due to clock and frequency drift. Figure 2 shows a Poll-Response-Final method of doing TWR and it also shows the formula used for calculation of TOF.



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

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

Figure 2: Asymmetric TWR TOF formula

2 IMPLEMENTATION OF RANGING

In Decawave’s two-way ranging demo, two units operate as a pair. One unit acts as a “Tag” initiating the ranging exchange and the other unit acts as an “Anchor” listening for the tag messages and performing two-way ranging exchanges with it. This is shown in Figure 3 below.

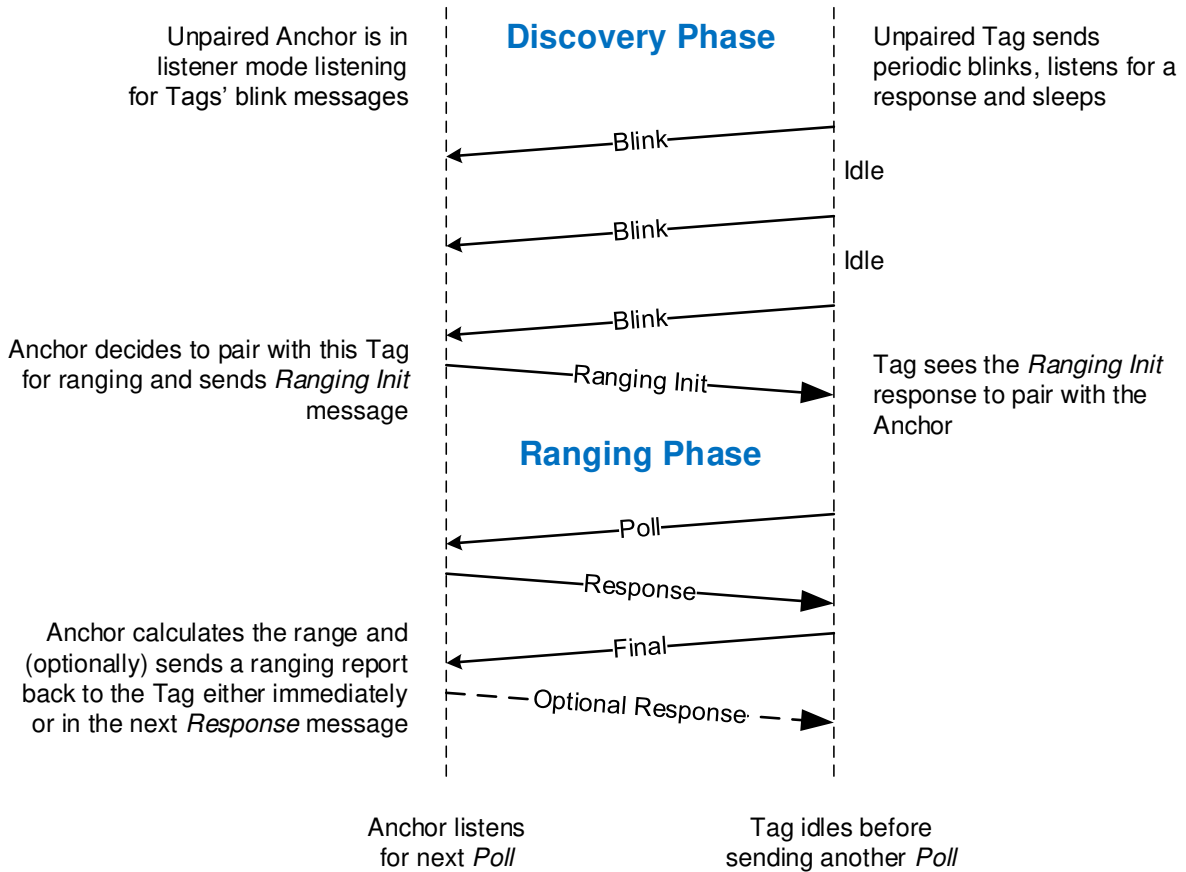


Figure 3: Discovery and Ranging phase message exchanges

2.1 Discovery phase

Initially the tag is in a discovery phase where it periodically sends a *Blink* message that contains its own address, and listens for a *Ranging Init* response from an anchor. If the tag does not get this response it sleeps for a period (default of 1 second) before blinking again. The anchor will initially listen for blinks, and when it receives a *Blink* message, the anchor will send a *Ranging Init* message to the tag, which will complete the *Discovery Phase* and enter the *Ranging Phase*.

2.2 Ranging phase

In the *Ranging Phase* the tag periodically performs two-way ranging exchanges with the anchor. Each two-way ranging exchange consists of the tag sending the *Poll message*, receiving the *Response* message and then sending the *Final* message. In the case where it is necessary for the tag to be aware of the range, the anchor may optionally send an immediate message to the tag with the calculated TOF or may wait until the next *Response* message to send a previous TOF. In either case, the tag can use the TOF to calculate the range.

For clarity, this optional message is not shown in the remainder of this document.

2.3 Messages used in ranging

Five messages are employed: two in the *Discovery Phase* (the *Blink* and *Ranging Init*) and three in the *Ranging Phase* (the *Poll*, the *Response*, and the *Final*), as shown in Figure 4. Although these follow IEEE message conventions, these are not standard RTLS messages, the reader is referred to ISO/IEC 24730-62 for details of standardised message formats for use in RTLS systems based on IEEE 802.15.4-2011 UWB. The formats of the messages used in the Decawave implementation are given in the following sections.

2.3.1 General ranging frame format

The general message format, shown at the top of Figure 4, is the IEEE 802.15.4 standard encoding for a data frame. The two byte *Frame Control* octets vary between the messages as some use 8-octet (64-bit) addresses and others 2-octet (16-bit) addresses. A single 16-bit PAN ID (value 0xDECA) is used for all the messages. The only exception is the *Blink* message which is described in 2.3.2 below. In a real deployment, the PAN ID might be negotiated as part of associating with the network or it might be an installation configured constant. The blink message follows the format defined in clause 5.2.2.7 *Multipurpose blink frame* of the *IEEE Std 802.15.4e™-2012 (Amendment to IEEE Std 802.15.4™-2011)*.

The sequence number octet is incremented modulo-256 for every frame sent, as per IEEE rules. The source and destination addresses are either 64-bit numbers programmed uniquely into each unit (during manufacture) or 16-bit addresses temporarily assigned. The 2-octet FCS is a CRC frame check sequence following the IEEE standard, (this can be generated automatically by the DW1000 IC and appended to the transmitted message).

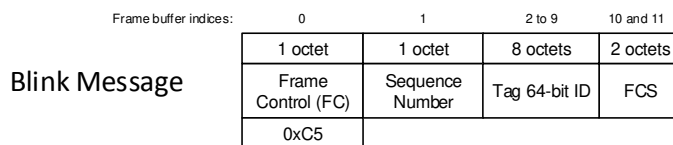
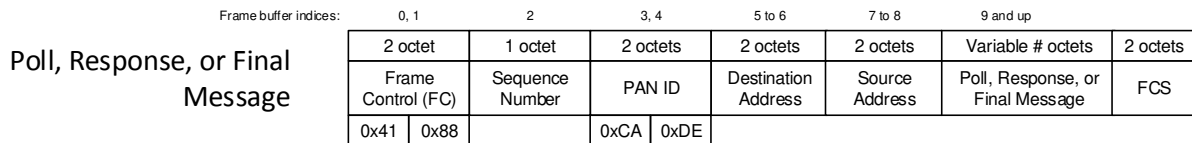
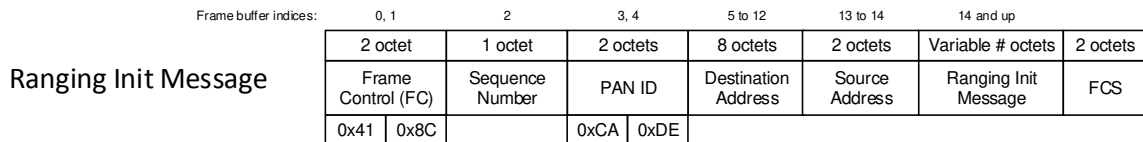
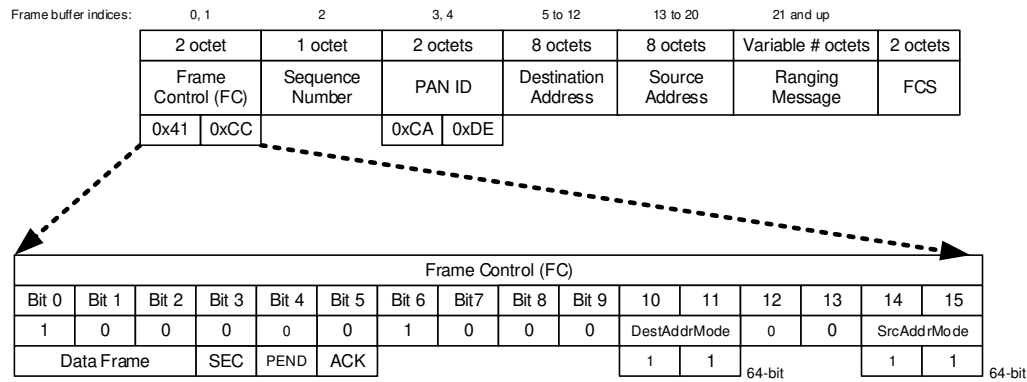


Figure 4: General ranging frame format

The content of the ranging message portion of the frame, (the "Variable # octets" part of the Poll, Response, Final and Report message shown above in Figure 4), defines which of the four ranging messages it is. We will also refer to this section of the message as the "application level payload". These are shown in Figure 5 and described in sections 2.3.3 to 2.3.6 below. In these, only the ranging message portion of the frame is shown and discussed. This data is encapsulated in the general ranging frame format of Figure 4 to form the complete ranging message in each case.

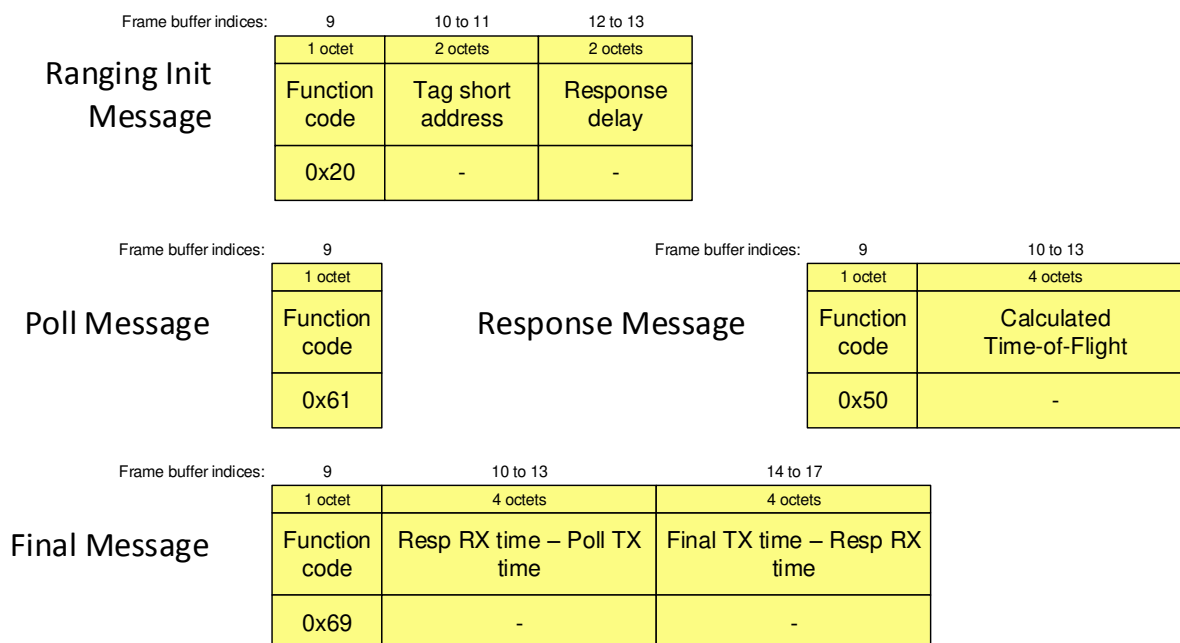


Figure 5: Ranging message encodings

2.3.2 Blink frame format

The *Blink message* frame format is used for sending of the tag *Blink* messages. The *Blink* frame is 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 this minimal blink is as shown in Figure 4.

2.3.3 Poll message

The *Poll* message is sent by the tag to initiate a single range measurement. For the poll message, the ranging message portion of the frame is a single octet, with the value: 0x61.

2.3.4 Response message

The *Response* message is sent by the anchor in response to a poll message from the tag. The *Response* message is 5 octets in length. Table 1 lists and describes the individual fields within the *Response* message.

Table 1: Fields within the Response message

Octet #'s	Value	Description
1	0x50	This octet value of 0x50 identifies the message as a <i>Response</i>
2 to 5	-	This four octet field is the anchor calculated time-of-flight, representing the estimated distance between the tag and the anchor. The time units are as defined in note 1 in 2.3.5 below.

2.3.5 Final message

The *Final* message is sent by the tag after receiving the anchor's response message. The *Final* message is 9 octets in length. Table 2 lists and describes the individual fields within the *Final* message.

Table 2: Fields within the Final message

Octet #'s	Value	Description
1	0x69	This octet identifies the message as the tag “ <i>Final</i> ” message
2 to 5	-	This four octet field is the difference between the <i>Final</i> TX timestamp and the <i>Response</i> RX timestamp. This value is pre-calculated by the Tag software and embedded in the message. The DW1000’s delayed send mechanism is used to ensure that the actual send time matches the value inserted here. The time units are defined in note 1 below.
6 to 9	-	This four octet field is the difference between the <i>Response</i> RX timestamp and the <i>Poll</i> TX timestamp. The time units are defined in note 1 below.

Note 1: The time units used are those defined in the IEEE standard and native to the DW1000, where the LSB represents 1/128 of the fundamental UWB frequency (499.2 MHz), or approximately 15.65 picoseconds.

2.3.6 Ranging Initiation message

Upon receiving the Blink message the unpaired anchor will send the Ranging Init message to the tag that has sent the Blink message.

The *Ranging Initiation* message is 5 octets in length. Table 3 lists and describes the individual fields within the *Ranging Initiation* message.

Table 3: Fields within the Ranging Initiation message

Octet #'s	Value	Description
1	0x20	This octet value of 0x20 identifies the message as a range report
2 to 3	-	This 16-bit field specifies the 16-bit address to be used by Tag for the ranging phase instead of its 64-bit address.
4 to 5	-	This 16-bit number gives the response time to be used in the following ranging exchange. The time units of this are ms.

2.4 TWR optimisation for power consumption

Minimising power consumption for a battery powered tag is an important consideration in order for the operational lifetime of the battery powered tag to be maximised. There are a number of factors to be considered here. These are described below with respect to the different stages in the ranging operation.

2.4.1 Discovery phase

To optimise power usage, while in this phase, the receiver on-time needs to be minimised. As the *Ranging Init* message is sent after a particular delay (shown in Figure 6), the tag should only turn on its receiver after this delay, and employ a receive timeout, to turn off the receiver if the ranging initiation frame does not arrive.

Figure 6 shows the time the tag spends in each of the states for the case when the anchor is using an 800 μ s response time for the *Ranging Init* message.

Note: The absolute minimum response time that can be achieved would be around 200 μ s, this is microprocessor dependent, it depends how quickly the microprocessor can see the completion of reception and start the transmission. However in the example in Figure 6, an 800 μ s delayed response time is used. This is because when the “Smart Tx Power” option is enabled for the 6.81 Mbps data rate, only 1 frame can be transmitted in 1 ms in order comply with ETSI and FCC regulations on TX power.

All times shown in Figure 6 are in microseconds (μs).

TWR DISCOVERY-PHASE TIMING PROFILE

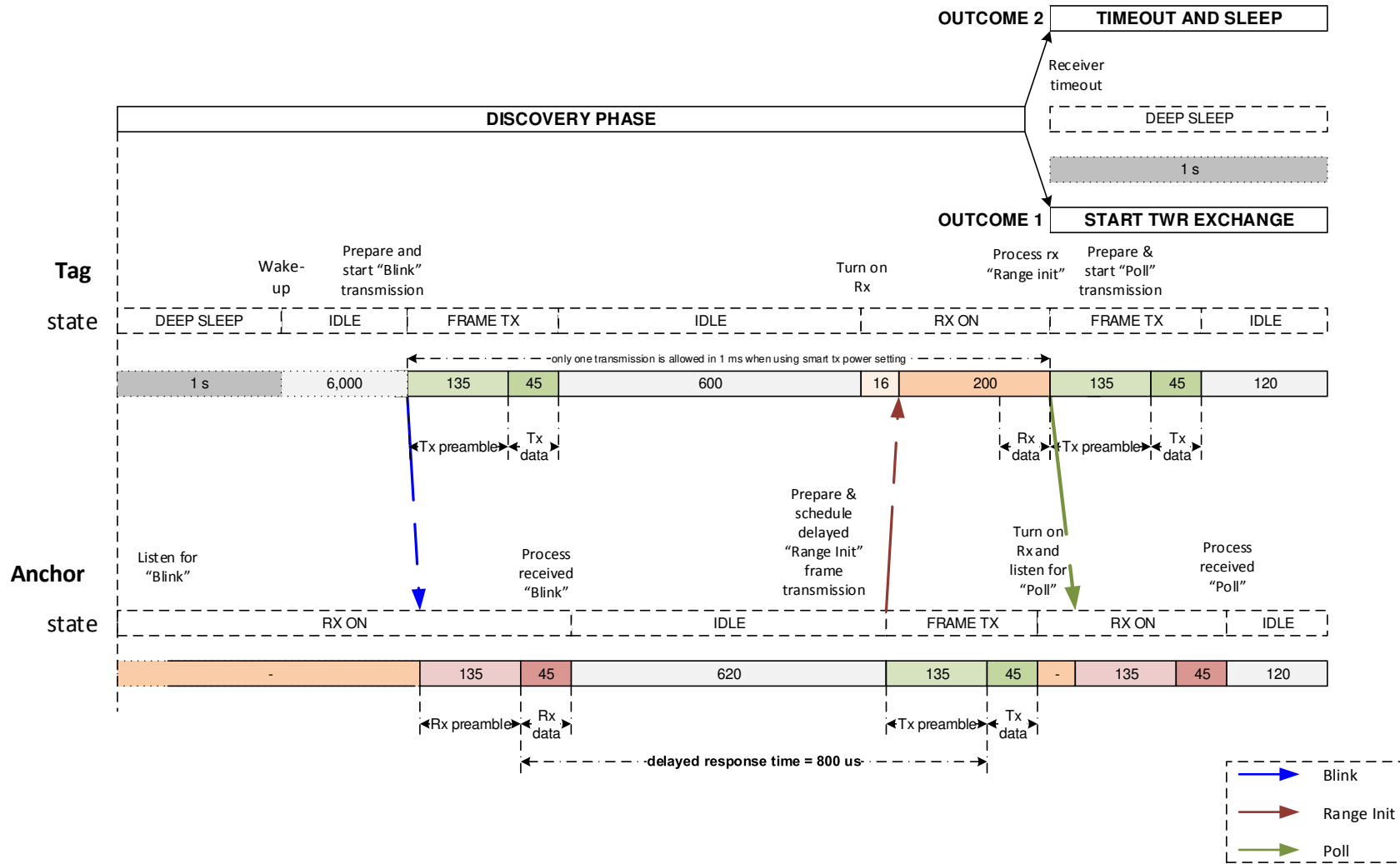


Figure 6: TWR discovery phase timing profile

2.4.2 Ranging phase

To save power, in this phase, the tag needs to complete these operations as quickly as possible, and then return to low power mode (Sleep state).

In this example, to reduce the transmission, and reception times, a preamble length of 128 is used together with shortened ranging messages (section 2.3 describes the message formats), and the highest data rate of 6.81 Mbps. As a result of using this configuration the total frame transmission time is about 180 μ s, and the reception time is about 215 μ s. The reception time is longer because of a 16 μ s receiver start-up delay, and due to the execution time of the leading edge detection search that is part of our receive time-stamping and takes up to 60 μ s after SFD detection.

To reduce the time spent in the idle state, the duration of SPI transactions also need to be minimised. This is limited by the microcontroller's SPI peripheral abilities. As DW1000 supports SPI speeds of up to 20 MHz the microcontroller should be configured to run at 20 MHz if possible (We have used an STM32 device which is limited to 18 MHz). Measures should be taken to send all the bytes of an individual SPI transaction back-to-back without any dead time between them (e.g. by employing DMA on the host processor if possible). As well as the physical speed of SPI operations the application also needs to make sure that the number of SPI read and write operations are minimised (i.e. only read / write the necessary registers for the required operation).

The short response times mean that 32-bit timestamp arithmetic can be used, i.e. since 2^{32} divided by the LSB of timestamps (128 x 499.2 MHz) is 67.2 ms. This further minimises the processor execution time and saves power.

Note: The absolute minimum response time that can be achieved would be around 200 μ s, this is microprocessor dependent, it depends how quickly the microprocessor can see the completion of reception and start the transmission. The anchor will respond immediately, but the tag will only send the *Final* 1 ms after the transmission of the *Poll*. This is because when the smart tx power option is enabled for a 6.81 Mbps data rate, only 1 frame can be transmitted in 1 ms to meet the ETSI / FCC regulations on transmitted power.

All times shown in Figure 7 are in microseconds (μs) unless otherwise stated.

TWR RANGING-PHASE TIMING PROFILE

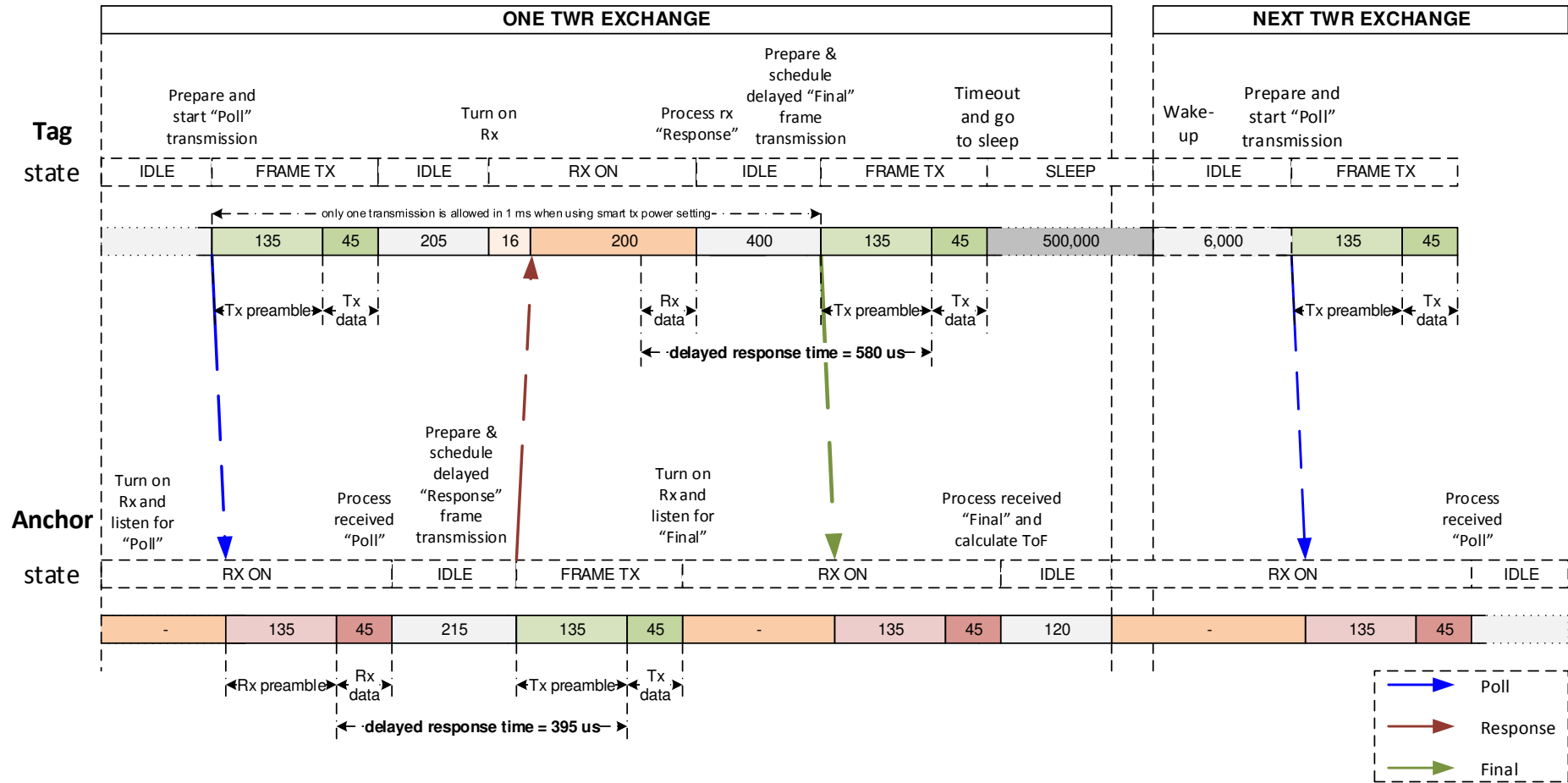


Figure 7: TWR ranging phase timing profile

3 CONCLUSION

This application note has given an overview of two way ranging as it is implemented by Decawave in its DecaRanging application. We have also outlined optimisations that can be applied to minimise power consumption for a battery powered device.

4 REFERENCES

Reference is made to the following documents in the course of this application note: -

Table 4: Table of References

Ref	Author	Version	Title
[1]	Decawave	Current	DW1000 Data Sheet
[2]	Decawave	Current	DW1000 User Manual
[3]	Decawave	Current	APS003 Real Time Location Systems
[4]	Decawave	Current	APS011 Sources of Error in DW1000 Based two-way ranging (TWR) Schemes

5 DOCUMENT HISTORY

Table 5: Document History

Revision	Date	Description
1.0	15 th December 2014	Initial release
1.1	31 st March 2015	Scheduled update
2.0	31 st December 2015	Scheduled update
2.1	30 th June 2016	Scheduled update
2.2	31 st March 2017	Scheduled update

6 MAJOR CHANGES

Revision 1.0

Page	Change Description
All	Initial release

Revision 1.1

Page	Change Description
All	Change Copyright notice to 2015
1	Change revision number to 1.1
11	Fix incorrect reference
13	Add 1.1 to revision table Add this table
14	Add new page

Revision 2.0

Page	Change Description
All	Updated references and page numbers
1	Change revision number to 2.0
All	Updated the text to refer to the asymmetric TWR method and related message formats and timings Update all diagrams to use the asymmetric TWR method.

Revision 2.1

Page	Change Description
All	Updated references and page numbers
All	Typographical corrections
1	Change revision number to 2.1
7	Repaired "Reference not found" error

Revision 2.2

Page	Change Description
All	Updated references and page numbers
All	Typographical corrections
1	Change revision number to 2.2
5	Modification to figure 3 to show optional response
5	Modification to explanatory text
13	Addition of v2.2 to revision history table
14	Addition of this table

7 ABOUT DECAWAVE

Decawave is a pioneering fabless semiconductor company whose flagship product, the DW1000, is a complete, single chip CMOS Ultra-Wideband IC based on the IEEE 802.15.4-2011 UWB standard. This device is the first in a family of parts that will operate at data rates of 110 kbps, 850 kbps and 6.8 Mbps.

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

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

Decawave Ltd
Adelaide Chambers
Peter Street
Dublin D08 T6YA
Ireland
t: +353 1 6975030
e: sales@decawave.com
w: www.decawave.com



Relación de documentos

<input type="checkbox"/> Memoria	83	páginas
<input checked="" type="checkbox"/> Anexos	224	páginas

La Almunia, a 28 de 11 de 2018

Firmado: Joaquín Pérez Sancho