APS019 APPLICATION NOTE

ISSUES TO CONSIDER WHEN PORTING THE DECAWAVE DECARANGING SOURCE CODE TO AN 8-BIT MCU

Version 1.1

This document is subject to change without notice

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

DecaWave's DW1000 is a single chip UWB transceiver, enabling the development of highly accurate, cost effective location solutions.

As part of customer development and evaluation support DecaWave supplies an evaluation kit, EVK1000, and provides example two-way ranging application (DecaRanging) software source code. The EVK1000 uses an ARM Cortex M3 based microcontroller (STM32F105RCT6) which is a 32-bit, little-endian machine

Customers may wish to use a different microcontroller to that used in the EVK1000, e.g. an 8-bit device, in their own design. This document outlines the areas to be considered by the developer when porting the DecaRanging application software to an 8-bit microcontroller.

This document should be read in conjunction with APS013 “DW1000 and two way ranging” which describes the two-way ranging technique and gives an overview of the DecaRanging application.

2 ASPECTS OF PORTING THE CODE

2.1 Introduction

Before starting to port to the desired microprocessor the developer should bear in mind some key characteristics of the chosen microprocessor. These include: -

- whether the machine is little-endian or big-endian
- whether it is an 8, 16 or 32 bit machine
- the memory architectures which may vary and affect the storage of different types of variables.

2.2 Consideration of little-endian and big-endian memory architectures

2.2.1 Determining which memory architecture is used

8-bit storage is used in DecaWave's example code to ensure portability of the code to other memory architectures. As already stated the MCU in the EVK1000 is a 32-bit, little-endian machine.

The example test code below prints out a message identifying the differences between little-endian and big-endian memory storage.

```
void test_endian(void)
{
	union e {
		unsigned long int	var;
		unsigned char		array[4];
	};

	e.array[0] = 0x0D;
	e.array[1] = 0x0C
	e.array[2] = 0x0B;
	e.array[3] = 0x0A;

	if ( e.var == 0x0A0B0C0D )
	{
		printf("Little-endian\n\r");
	}
	else if (e.var == 0x0D0C0B0A)
	{
		printf("Big-endian\n\r");
	}
	else
	{
		printf("Middle-endian or unknown storage type. Variable= %x", e.var);
	}
}
```

In this example, in the case of little-endian architecture the variable *var* will be `0x0A0B0C0D`, while for big-endian the variable *var* will be `0x0D0C0B0A`.

2.2.2 Networking concerns

Since it is desirable for the same C code to work on different MCU types which can then interwork, care needs to be taken with the communication channel byte order (i.e. the order of the bytes sent over the air needs to be the same in all of the architectures).

In any system where MCUs with different architectures must be used together and where different payloads are sent and received e.g. 32-bit values of timestamps and calculated differences between timestamps then it is important that all nodes in the system process information in the same way, so that at the application level the order of bytes is the same.

A common method is to write specific **transform functions** to change from one endian type to the other when sending and receiving data. These transforms are generally denoted as follows: -

- For sending data from the host to the network: -

`htnol()`: host-to-network-long transform function for 32-bit values
`htons()`: host-to-network-short transform function for 16-bit values

- For receiving data from the network to the host: -

`ntohl()`: network-to-host-long transform function for 32-bit values
`ntohs()`: network-to-host-short transform function for 16-bit values

Another way is to avoid the use of 16-bit or 32-bit values directly in communication buffers and always use unsigned char arrays for network data payload sending / receiving.

In all 802.15.4 communications including the DecaRanging application, data transferred in the payload always starts with the least significant bit first in time order.

2.3 Compiler differences

2.3.1 Introduction

In porting the code, consideration needs to be given to potential differences between compilers.

The EVK1000 DecaRanging code is compiled using the `gcc` compiler for ARM Cortex M3. Other MCUs will require the use of different compilers e.g. for the C8051 MCU one possible compiler is the `C51` compiler from Keil.

2.3.2 Pointer types

The `C51` compiler uses different pointer types for code and data memory storage access. Without special keywords present in function prototypes, the `C51` compiler assumes that pointers are for data memory. Where it is intended to point to code rather than data, e.g. a function pointer, it is necessary to add the keyword “code” to instruct the compiler that a pointer is to be treated as a “function pointer” pointing to code memory. Please refer to the `C51` compiler datasheet for additional information if necessary:

http://www.keil.com/support/man/docs/c51/c51_le_ptrconversions.htm

2.3.3 Variable truncation and optimization

If the MCU has an 8-bit core, the compiler may also truncate (optimize) variables when performing operations, especially when performing shift operations. To prevent this, the developer needs to add C-casting operations to ensure the compiler operates as intended. An example of incorrect (non-portable) and correct (portable) implementations of a shift operation is shown below -

```
	unsigned char  abc;
	unsigned long int def;

	abc = 0x01;

	// incorrect implementation: in the line below we expect
def=0x01000000, but for C51 it will be zero

	def = abc<<24;

	// correct implementation: in next line result will be as expected,
```

```
0x01000000

      def = ((unsigned long int)abc)<<24;
```

2.4 *Software Architecture & Hardware platform abstraction layer*

2.4.1 Overview

The source code of the DecaRanging application, as provided by DecaWave, consists of three layers:

1) Application layer [**APPLICATION**]
2) Platform independent driver layer [**DECADRIVER**]
3) Platform dependent driver layer [**PLATFORM**]

The **APPLICATION** layer and **DECADRIVER** layer software are described in references [3] and [4] respectively. Please refer to these documents for more details.

The **PLATFORM** layer provides the software interface from the **APPLICATION** and **DECADRIVER** layers to the target specific hardware. When porting, the developer must ensure that the **PLATFORM** layer works correctly so that the application operates as expected.

The most important elements of the **PLATFORM** layer are as follows: -

- Clocking functionality
- SPI interface handling
- Interrupt handling
- Mutex mechanism

2.4.2 Clocking

For the DecaRanging application the clocking functionality must provide a "Sleeping" function, i.e. pausing functionality with CLOCK_PER_SEC resolution. This is the *sleep_ms* function in *deca_sleep.c*.

For the `gcc` compiler this functionality is provided through "`time.h`" libraries and in respect of hardware through re-entrant functions declared in "`syscalls.h`".

For example, in the case of the C51 compiler, the developer needs to create a hardware dependent clock system, which consists of the necessary clock functions, for example: -

- oscillator HW initialisation function
- system timer initialisation and its call-back function
- `sleep()` function

2.4.3 SPI interface

The DW1000 transceiver is controlled through its slave SPI interface by the external microcontroller. This slave SPI interface has a maximum SPI clock rate of 20 MHz.

The developer must implement specific code for its hardware. For SPI interface, it includes *writetospi* and *readfromspi* functions from *deca_spi.c*, *spi_set_rate_low* and *spi_set_rate_high* from *port.c*. The SPI initialisation can be handled by the *spi_peripheral_init* function in *port.c*, or in a custom function created by the developer.

In battery powered applications, power consumption is one of the main design concerns, thus it is desirable to have the SPI rate as fast as possible. This minimizes the amount of time taken per SPI transaction thereby minimizing the power consumption associated with the transfer.

Some SPI implementations have an inter-byte spacing between each of the transferred bytes in a transaction, as shown in Figure 1. Minimizing this inter-byte spacing ensures that the resultant SPI transfer rate is as close as possible to the SPI clock rate (e.g. 20 MHz). This has a bearing on the choice of MCU that is suitable for particular application. Using DMA for large SPI transfers will also help this, as shown in Figure 2.

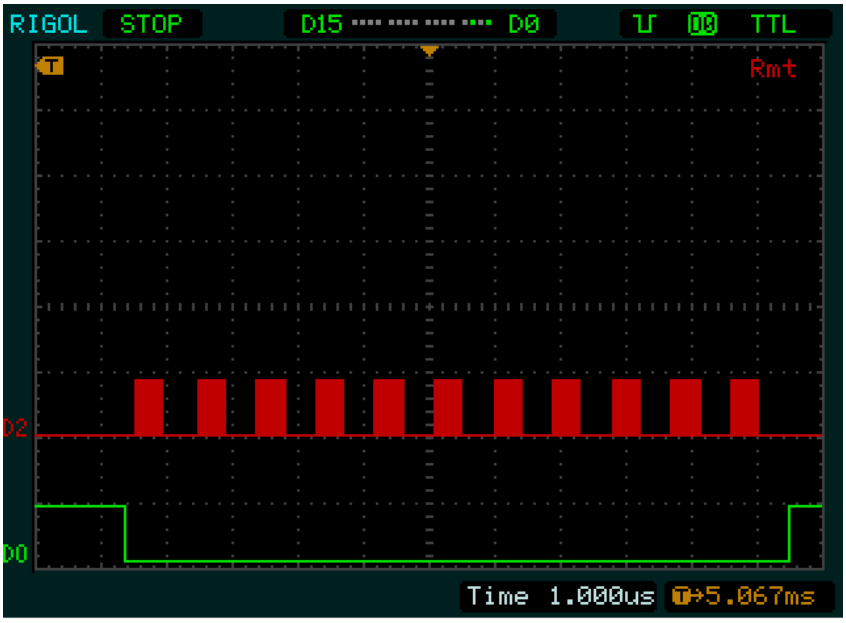

Figure 1: 18 MHz SPI transaction using Cortex M3 without DMA

Figure 1 shows an 11-byte SPI transaction using the Cortex M3 without DMA support resulting in inter-byte spacing in the transaction. In this example, for an SPI clock of 18 MHz, the effective data rate is approximately 9 Mbit/s.

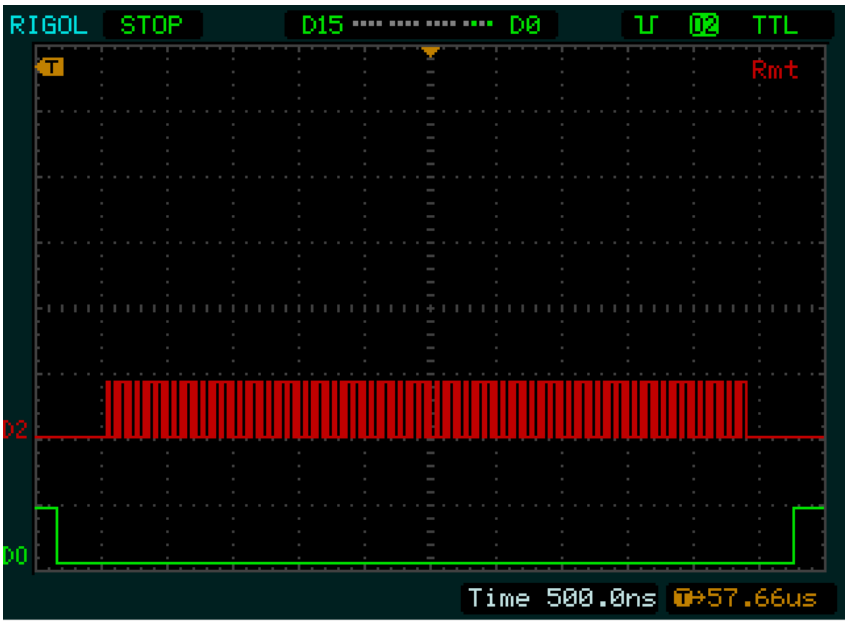

Figure 2: 18MHz SPI transaction using Cortex M3 with DMA

Figure 2 shows an 11-byte SPI transaction using the Cortex M3 with DMA support. There is no inter-byte spacing so with an SPI clock of 18 MHz the effective data rate is approximately 16 Mbit/s.

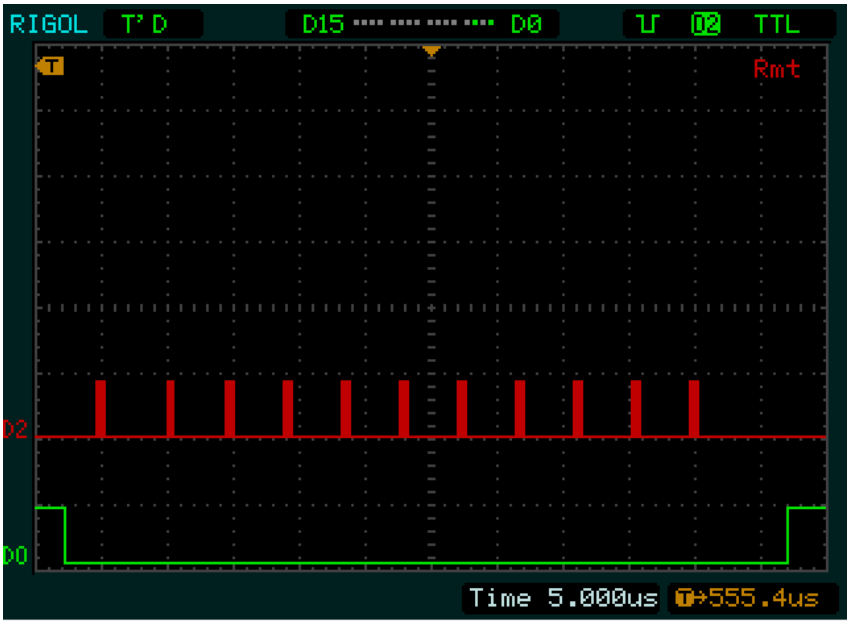

Figure 3: 12MHz SPI transaction on C8051 with no DMA & no IRQ

Figure 3 shows an 11-byte SPI transaction using the C8051F381 with an SPI clock of 12 MHz, with no DMA support and no IRQ. The effective data rate is approximately 1.6 Mbit/s. Here the inter-byte spacing in the transaction reduces the effective data rate of the SPI interface from a possible 12 Mbit/s by a factor of almost ten times. Depending on the intended application this may be an issue, which may mean that the particular 8-bit MCU is not suitable.

2.4.4 External IRQ handling

DW1000 uses its interrupt request (IRQ) pin to indicate to the MCU, and thus the **APPLICATION**, that some event has occurred. Using interrupts is a well-known method for rapidly and efficiently processing events.

The default active level of the DW1000 IRQ output is high, so when the DW1000 has no events, it will drive its IRQ pin low. If the MCU puts the DW1000 into DEEP SLEEP mode, the DW1000 will release its IRQ pin allowing it to float. This may cause unnecessary MCU interrupts. To avoid this it is highly recommended to use an external pull-down resistor connected to the MCU IRQ pin. See [2].

2.4.5 Mutex mechanism

When porting the DecaRanging application to another MCU, it is important to protect the handling of interrupts from DW1000 IRQ line to prevent concurrent access to the MCU SPI peripheral block by different independent sources (i.e. threads, IRQ, etc.).

Example `mutex` functions are included in the DecaRanging source code; the developer will need to port these to their own implementation / system in *deca_mutex.c*.

3 CONCLUSION

This application note describes the main areas a developer needs to consider when porting DecaWave's example DecaRanging source code to a microprocessor of their choice.

Through the careful design and porting of **PLATFORM** layer functions, the example application can be easily ported to various microcontrollers with minimum debug.

4 REFERENCES

Reference is made to the following documents in the course of this application note: -

Table 1: Table of References

Ref	Author	Version	Title
[1]	DecaWave	Current	DW1000 Data Sheet
[2]	DecaWave	Current	DW1000 User Manual
[3]	DecaWave	Current	DecaWave API Guide
[4]	DecaWave	Current	APS013 DW1000 and two-way ranging

5 DOCUMENT HISTORY

Table 2: Document History

Revision	Date	Description
1.0	30th June 2015	Initial release
1.1	31-August-2018	New logo update

6 MAJOR CHANGES

v1.0

Page	Change Description
All	Initial release

v1.1

Page	Change Description
All	Logo update
6	Updated SPI section with additional implementations required

7 FURTHER INFORMATION

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

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