M252 Porting Google Protocol Buffers

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code demonstrates how to use Nanopb to implement protocol buffers in M252. Nanopb is an ANSI-C implementation of Google Protocol Buffers data format.
BSP Version	M251/M252/M254/M256/M258 Series CMSIS BSP V3.01.002
Hardware	NuMaker-M252SD Ver1.2

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

Protocol Buffers (Protobuf) is a free and open source cross-platform library used to serialize structured data. It is useful in developing programs to communicate with each other over a network or for storing data. The method involves an interface description language that describes the structure of some data and a program that generates source code from that description for generating or parsing a stream of bytes that represents the structured data.

The M252 provides sample code to demonstrate how to actually make protocol buffer functionality. Note that this sample code also applies to the M251/M254/M256/M258 series.

The sample code uses Petteri Aimonen's Nanopb open source library. Nanopb download address is https://jpa.kapsi.fi/nanopb/download/ .

1.1 How to Use the Open Source Code Nanopb

Nanopb is an ANSI-C library for encoding and decoding messages in Google's Protocol Buffers format with minimal requirements for RAM and code space. It is primarily suitable for 32-bit microcontrollers, but is also fit for other embedded systems with tight (<10 kB ROM, <1 kB RAM) memory constraints.

nanopb-0.4.5-windows-x86 is used here. First, use Sample Template to create a Protobuf project, and then add the seven files, *pb.h*, *pb_common.c*, *pb_common.h*, *pb_decode.c*, *pb_decode.h*, *pb_encode.c*, and *pb_encode.h* under the nanopb-0.4.5-windows-x86 directory copied to the project folder, as shown in Figure 1-1 Project Folder

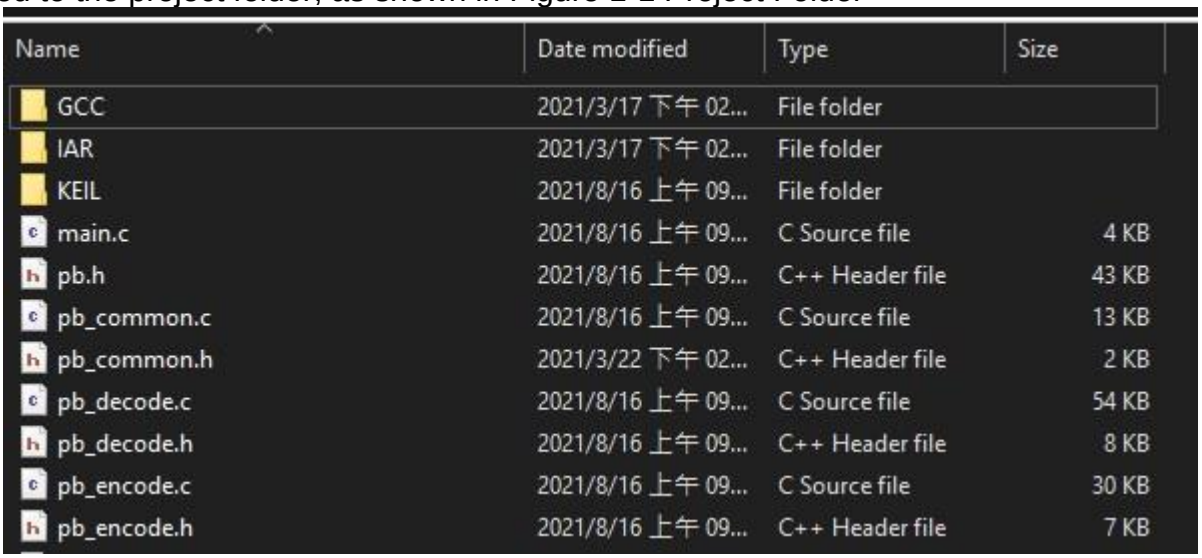


Figure 1-1 Project Folder Contents

The core of Protobuf is a .proto file. Users can customize a .proto to create the protocol data, and then use the protoc tool to compile and generate C code. There are two files: a header file and a source file. The protoc tool is in the generator-bin folder of nanopb-0.4.5-windows-x86 that has been downloaded, as shown in Figure 1-2 protoc Tools in generator-bin Folder

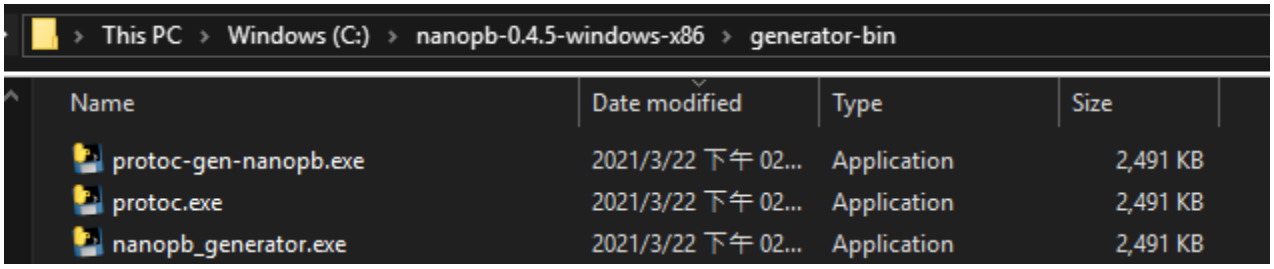


Figure 1-2 protoc Tools in generator-bin Folder

In order to be able to use the protoc tool anywhere on the command line, users need to add the path where the *protoc.exe* is located to the environment variable. Then, create a test.proto file in the Protobuf folder.

```
syntax = "proto2";

message Student
{
 required uint32 num      = 1;
 required uint32 py_score = 2;
 required uint32 c_score  = 3;
}
```

Enter the command line window, and on the command line cd to the directory where test.proto is located. Enter the following command to compile test.proto: protoc --nanopb_out=. test.proto to generate *test.pb.c* and *test.pb.h* files as shown in Figure 1-3 Command Line Window and Figure 1-4 Project Folder Contents

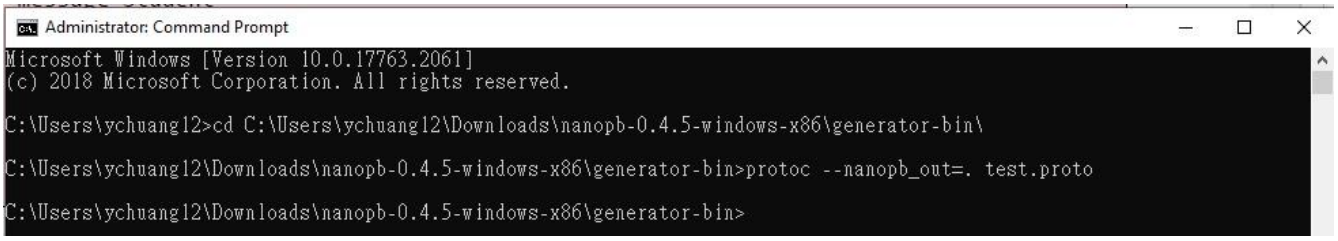
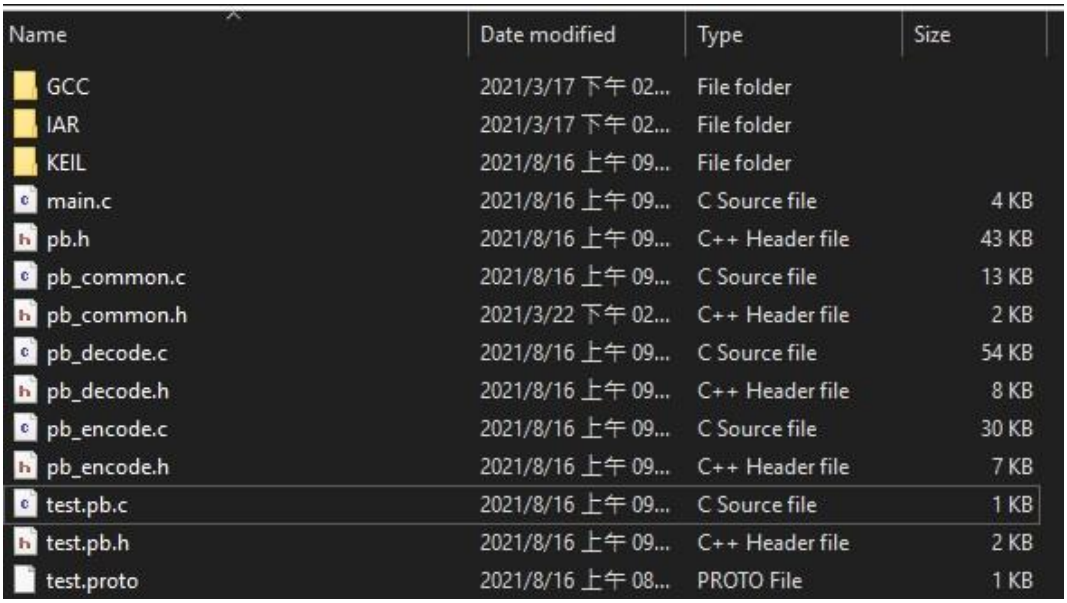


Figure 1-3 Command Line Window



Name	Date modified	Type	Size
GCC	2021/3/17 下午 02...	File folder	
IAR	2021/3/17 下午 02...	File folder	
KEIL	2021/8/16 上午 09...	File folder	
main.c	2021/8/16 上午 09...	C Source file	4 KB
pb.h	2021/8/16 上午 09...	C++ Header file	43 KB
pb_common.c	2021/8/16 上午 09...	C Source file	13 KB
pb_common.h	2021/3/22 下午 02...	C++ Header file	2 KB
pb_decode.c	2021/8/16 上午 09...	C Source file	54 KB
pb_decode.h	2021/8/16 上午 09...	C++ Header file	8 KB
pb_encode.c	2021/8/16 上午 09...	C Source file	30 KB
pb_encode.h	2021/8/16 上午 09...	C++ Header file	7 KB
test.pb.c	2021/8/16 上午 09...	C Source file	1 KB
test.pb.h	2021/8/16 上午 09...	C++ Header file	2 KB
test.proto	2021/8/16 上午 08...	PROTO File	1 KB

Figure 1-4 Project Folder Contents

Add files from the Protobuf folder to the mdk environment as shown in Figure 1-5 Compiler Environment and add the search path of the header file.

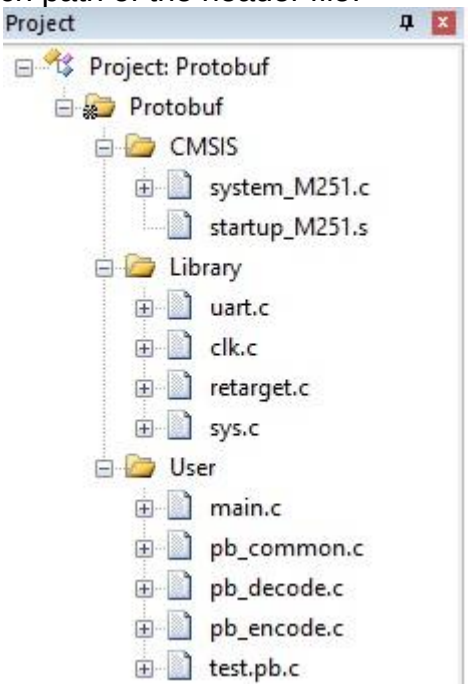


Figure 1-5 Compiler Environment

Demo is very simple. Encoding means constructing a protocol data structure, calling the encoding to write data into the buffer; decoding is the other way around, reading data from the buffer and decoding it into the structure. Among them, the encoding-related functions are in pb_encode.c, and the decoding-related functions are in pb_decode.c.

1.2 Demo Result

The demo result is shown in Figure 1-6 Demo Result

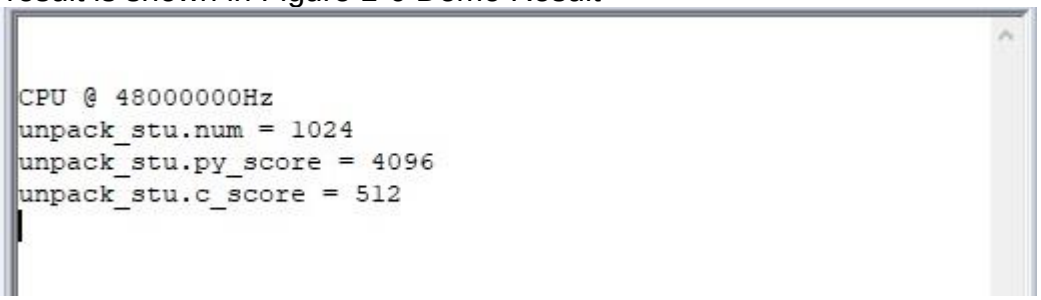


Figure 1-6 Demo Result

2 Code Description

Initialize pack_stu at the beginning, transmit it through o_stream after encoding, and then decode it through i_stream to receive the data to write unpack_stu. The comparison result is shown in Figure 1-6 Demo Result

```
    /* pack_stu and unpack_stu is Student type*/
    Student pack_stu = {0};
    Student unpack_stu = {0};
    /* pack_stu initialization */
    pack_stu.num  = 1024;
    pack_stu.py_score = 4096;
    pack_stu.c_score = 512;
    /* o_stream initialization */
    o_stream = pb_ostream_from_buffer(buffer, sizeof(buffer));
    /* encode all pack_stu fields from Student into o_stream */
    pb_encode(&o_stream, Student_fields, &pack_stu);
    /* i_stream initialization */
    i_stream = pb_istream_from_buffer(buffer, sizeof(buffer));
    /* decode all fields from i_stream into unpack_stu */
    pb_decode(&i_stream, Student_fields, &unpack_stu);
    /* print unpack_stu result */
    printf("unpack_stu.num = %d\n", unpack_stu.num);
    printf("unpack_stu.py_score = %d\n", unpack_stu.py_score);
    printf("unpack_stu.c_score = %d\n", unpack_stu.c_score);
```

Encode all fields:

```
bool checkreturn pb_encode(pb_ostream_t *stream, const pb_msgdesc_t *fields, const void
*src_struct)
{
    pb_field_iter_t iter;

    if (!pb_field_iter_begin_const(&iter, fields, src_struct))
        return true; /* Empty message type */

    do
    {
        if (PB_LTYPE(iter.type) == PB_LTYPE_EXTENSION)
        {
            /* Special case for the extension field placeholder */
            if (!encode_extension_field(stream, &iter))
                return false;
        }
```

```
        else
        {
            /* Regular field */
            if (!encode_field(stream, &iter))
                return false;
        }
    } while (pb_field_iter_next(&iter));

    return true;
}
```

Decode all fields:

```
bool checkreturn pb_decode(pb_istream_t *stream, const pb_msgdesc_t *fields, void
*dest_struct)
{
    bool status;

    status = pb_decode_inner(stream, fields, dest_struct, 0);

#ifdef PB_ENABLE_MALLOC
    if (!status)
        pb_release(fields, dest_struct);
#endif

    return status;
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251/M252/M254/M256/M258 Series CMSIS BSP V3.01.002
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M252SD Ver1.2
- Pin Connect

UART_TX (PB.13)	↔	RX
UART_RX (PB.12)	↔	TX
M252		PC

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown below.

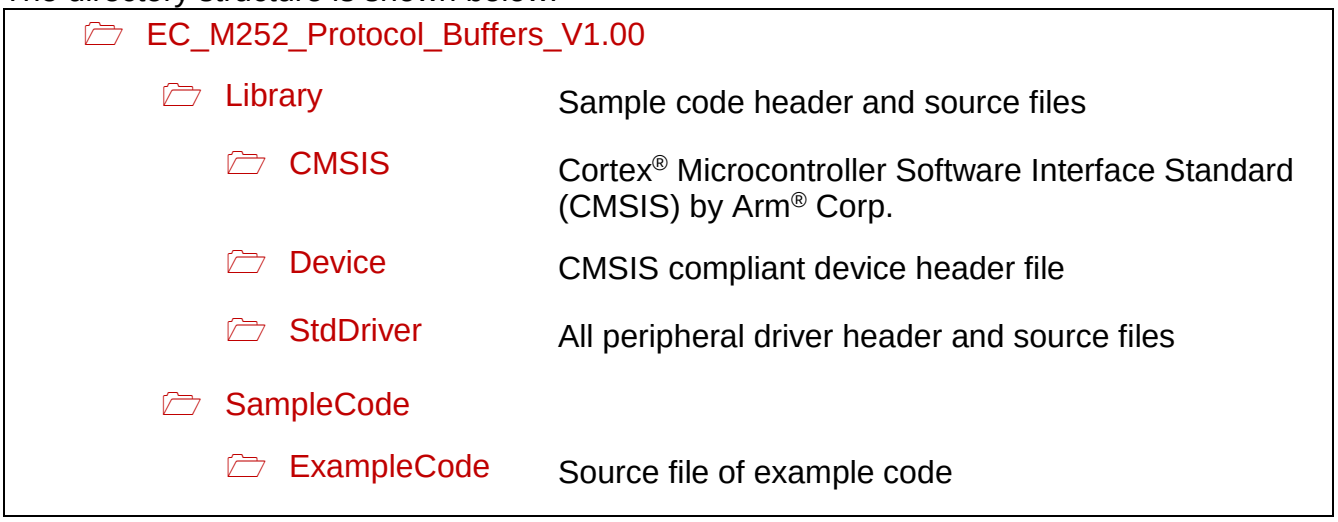


Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M252_Protocol_Buffers.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6 Revision History

Date	Revision	Description
2021.09.24	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.