

Libserialport manual for IP2

F. J. Barzilaij

April 27, 2025

1 Introduction

This manual provides instructions on how to download and use libserialport.h, developed by sigrok. Currently this manual only contains steps for a windows device. The author does not claim any credit for the libserialport library. The manual begins by explaining how to install a C compiler and set up the required environment. It then details how to install the external library using MSYS2 and other compilers. Finally, it demonstrates how to successfully run a program with the external library using CMake and the terminal. The author used MSYS2 with MinGW64, CLion, Visual Studio Code, and a Windows 11 laptop. Other combinations may work differently and are not tested. Please carefully read the README from sigrok if you plan to use a different setup.

libserialport is communicates with Zigbee modules via COM ports. Although COM ports can also be accessed via built-in Windows commands (via windows.h) or other libraries, libserialport is preferred because it is cross-platform (available via Homebrew on macOS and easily installable on Linux). Another reason for choosing libserialport is its similarity to Python's pyserial library in terms of use and naming, which will be used later in the Bachelor course.

2 C Compiler Installation

This chapter explains how to install a C compiler with all required packages and path definitions. The author has chosen to document the installation using MSYS2. If you use a different compiler, please refer to its README on the github page of Sigrok for installation instructions.

MSYS2 can be downloaded from <https://www.msys2.org/>. Download the file `msys2-x86_64-20250221.exe`. After launching the installer, specify the installation folder (or leave it as default) and click through the steps. After installation, a folder called `msys64` will be created containing many `.exe` files. Two of these are important:

1. **msys2.exe** — the standard MSYS2 terminal
2. **mingw64.exe** — the terminal for the MinGW64 distribution

It is important to use the correct terminal when required. To update the system, open the MSYS2 terminal and run:

```
1 $ pacman -Syu
```

After running this line the terminal should have been closed and needs to be reopened. Then run the following code.

```
1 $ pacman -Su
```

Now several packages are required to be installed, These have to do with the gcc, cmake, and installation of libserialport.

```
1 $ pacman -S autoconf automake-wrapper cmake git libtool make  
mingw-w64-x86_64-gcc
```

After installing these packages, the compiler's paths must be added to your environment variables (either User or System, depending on your installation). Add:

```
1 C:\[YOUR_ADDITIONAL_ADDRESS]\msys64\mingw64\bin  
2 C:\[YOUR_ADDITIONAL_ADDRESS]\msys64\mingw64\x86_64-w64-mingw32\bin
```

3 External Library Installation

This chapter explains how to install libserialport. There are several ways to install it; this manual focuses on using the MSYS2 environment. If you are using a different environment, please refer to the README file from sigrok.

To install the libserialport library:

1. Open the MSYS2 terminal (msys2.exe).
2. Change directory to where you want to install the library:

```
1 $ cd /mingw64
```

3. Clone the library using git:

```
1 $ git clone https://github.com/sigrokproject/libserialport.git
```

4. After cloning, close the MSYS2 terminal and open the mingw64.exe terminal. Using the wrong terminal will cause compilation errors. Then, run the following commands one by one, allowing each to complete before moving to the next:

```
1 $ ./autogen.sh
2 $ ./configure
3 $ make
4 $ make install
```

If all steps have been successful then libserialport is installed correctly on your device. However, some things still need to be done in order to compile successfully.

4 Compiling with External Libraries

Using external libraries requires extra steps when compiling your code. The easiest (but least scalable) method is using the terminal directly (such as the terminal in Visual Studio Code or CLion). The following terminal code is used where *-lserialport* indicates the external library.

```
1 gcc main.c additional.c -o main.exe -lserialport
```

If the terminal gives an output of error that *lserialport* is undefined. Then the path of such files should be extended like:

```
1 gcc main.c additional.c -o main.exe -lserialport
  -LC:[YOUR_ADDITIONAL_ADDRESS]/msys64/mingw64/lib
  -IC:[YOUR_ADDITIONAL_ADDRESS]/msys64/mingw64/include
```

A better long-term solution is using a CMakeLists.txt file, especially for larger projects. This works both with Visual Studio Code and CLion, but CLion supports this much better as it can auto build and help the user further develop the CMakeLists as more files are added. An example of how such a code would look is shown below.

```
1 cmake_minimum_required(VERSION 3.28) #can be different but best if
   specified
2 project([PROJECT_NAME] C)
3
4 set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} -mconsole")
5 # Specify the path to the include directory where libserialport.h is
   located
6 include_directories("/mingw64/include")
7
8 # Add the path to the library directory where libserialport.a is located
9 link_directories("/mingw64/lib")
10
11 # Add the executable target
12 add_executable(CSpace main.c additionnal.c)
13
14 # Link the libserialport library to your executable
15 target_link_libraries(CSpace libserialport)
```

(Note: *additional.c* should have an accompanying header file but doesn't need to be listed separately in CMake.) On some laptops, CMake can sometimes cause stack overflow errors when working with external libraries. If issues arise, try compiling first via terminal commands to verify the setup. If it works via terminal commands refer to the troubleshoot section or contact a TA.

5 Libserialport useful parameters and Functions

Libserialport contains a lot of parameters that the user can and should define, as well as a lot of built in functions. This section will discuss the parameters and built-in functions that will be used during this project if it is chosen to use this external library. Do note that you are not limited to only use these parameters and built-in functions, you are free to use anything from the library.

5.1 Parameters

UART communication settings must be properly configured to match those defined on the Zigbee device. These settings need to be explicitly set in your C program to ensure correct interpretation of received data. There are five different parameters that should be defined by the following function

- **sp_set_baudrate([port], [baudrate])** – Sets the baud rate for the UART communication. The baud rate must match the value defined on the Zigbee module (Can be determined from XCTU).
- **sp_set_parity([port], [Paritybits_amount])** – Sets the parity mode (none, even, odd). The correct parity setting must be defined according to the IP2 manual (Can be determined from XCTU).
- **sp_set_stopbits([port], [Stopbits_amount])** – Sets the number of stop bits. This must match the settings from the Zigbee device documentation (Can be determined from XCTU).
- **sp_set_flowcontrol([port], [Flow_control_number])** – This function defines the type of flow control that is used by the UART. The UART does not use any flow control and therefore the value 0 is assigned (also defined as SP_FLOWCONTROL_NONE).
- **sp_set_bits** – Sets the number of data bits for UART communication. This value must match the Zigbee device's configuration, as specified in the IP2 manual (can be determined from XCTU).

5.2 Functions

The external library contains a lot of useful functions that can be used during the project. This manual will discuss 10 of them.

- **sp_open([port], SP_MODE_READ_WRITE)** – Opens the specified COM port in read/write mode. Always verify the COM port connected to the Zigbee module, as it can change after device reboots.
- **sp_close([port])** – Closes an open COM port. Always close ports when your program finishes using them.
- **sp_free_port([port])** – Frees the memory allocated for the port after it has been closed.

- **sp_get_port_by_name([portname_string], [port_pointer])** – Retrieves the pointer to a specific COM port given its name (e.g., "COM6").
- **sp_list_ports([list_pointer])** – Lists all active COM ports on the device. (Optional, but useful to make your program more adaptable. will change the struct by adding ****[list_pointer]** next to ***[port]**)
- **sp_get_port_name([port])** – Returns the port name string (e.g., "COM6") for a given port.
- **sp_get_port_description([port])** – Returns the description of the obtained port as a string. can be different per device and can be used to select specific COM ports.
- **sp_free_port_list([port_list])** – Frees the memory allocated for the port list, and should be used before program exit.
- **sp_nonblocking_write([port], [send_byte_pointer], [transfers])** – Starts writing data immediately to the specified port. [port] means to which port you will send the information. [send_byte_pointer] is the address of the information that will be send. [transfers] specifies the amount of bytes you will send.
- **sp_blocking_read_next([port], [receiving_byte_pointer], [transfers], [timeout])** – Waits until a specified number of new bytes are received or a timeout occurs. [port] is the COM port which receives the information from the Zigbee. [receiving_byte_pointer] is the address of the received byte. [transfers] is the amount of bytes that should be received. [timeout] is the time (in microseconds) for which the program will stay blocked, specify the timeout long enough for data to be received.

6 Final Notes

This manual and the use of this external library is first added to the project IP2 in the academic year of 2024/2025 as an alternative to the already used windows version of communicating with COM ports. As this is relatively new the author might have missed some things for specific computers and has no manual as of now to handle all errors that you might face when trying to use Libserialport for the first time. Some errors can be solved by consulting Appendix B. Please contact the author of this manual via Teams chat or simply contact him during your lab session in Tellegen.

In later version of this document a step by step installation method for Macbook and Linux will be added. Please consult the README file from Sigrok for now and if you face any difficulties. Just message the author via Teams chat.

A Example

```
1 #include <stdio.h>
2 #include <libserialport.h>
3
4 #define BAUDRATE 9600
5 #define TIMEOUT 50000
6
7 int main(void){
8     struct sp_port *port;
9
10    /* Opening the COMPORT. First function uses the
11       portname to then get the pointer of the port that
12       will be used.
13       * Then the second function opens the port in read
14       and write mode (ALWAYS CLOSE AT THE END OF THE
15       PROGRAM!)
16       * */
17    sp_get_port_by_name("COM6", &port);
18    sp_open(port, SP_MODE_READ_WRITE);
19
20    // Configure port settings (baudrate defined above,
21    // no parity, 1 stop, no flowcontrol, 8 data bits)
22    sp_set_baudrate(port, BAUDRATE);
23    sp_set_parity(port, SP_PARITY_NONE);
24    sp_set_stopbits(port, 1);
25    sp_set_flowcontrol(port, SP_FLOWCONTROL_NONE);
26    sp_set_bits(port, 8);
27
28    unsigned char send_byte = 0b11110111;
29    unsigned char recv_byte;
30
31    // Send data
32    printf("Sending: %d", send_byte); //lets the user
33    // know what is sent, helpful for debugging
34    sp_nonblocking_write(port, &send_byte, 1);
35
36    // Wait for response
37    int bytes_read = sp_blocking_read_next(port,
38    &recv_byte, 1, TIMEOUT);
39    if (bytes_read == 1) {
40        printf("Received: %d\n", recv_byte);
41    } else {
42        printf("Error reading from port.\n");
43    }
44 }
```

```
37
38     // Close the port (unreachable in an infinite loop,
        but good practice)
39     sp_close(port);
40     sp_free_port(port);
41
42     return 0;
43 }
```

main.c

B Troubleshoot

This section is still being worked on and contains some troubleshooting options. Not all have been verified to work

B.1 CMake error Stack Overflow

This error has not yet been solved but some steps can be taken after looking at different similar problems and fixes.

1. Making CMakeLists.txt more adaptable

```
1 find_library(SERIALPORT_LIBRARY serialport PATHS "/mingw64/lib")
2 find_path(SERIALPORT_INCLUDE_DIR libserialport.h PATHS
3           "/mingw64/include")
4 include_directories(${SERIALPORT_INCLUDE_DIR})
5 link_directories("/mingw64/lib")
6
7 add_executable(CSpace main.c additional.c)
8
9 target_link_libraries(CSpace ${SERIALPORT_LIBRARY})
```

2. Changing the overflow bound (highly skeptical that this actually solves the problem).

```
1 set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS}
   -Wl,--stack,268435456")
```