Muse: Timesync

Application Note

Revision History

Date	Revision	Author	Description
23/09/2024	1.0	RB	First draft

Table of Contents

1 Introduction

This application note aims to deepen the technical aspects related to the time synchronization of one or more devices (i.e., peripherals) simultaneously connected to the same master node (i.e., central device).

2 Time Synchronization

2.1 Requirements and Constraints

The proposed time synchronization procedure can be executed via Bluetooth or Serial (USB) connections, and requires:

- Availability of at least one device (e.g., Muse or Mitch) to be connected and synchronized.
- Availability of a device (e.g., desktop PC, laptop or smartphone) that operates as master node, that supports Bluetooth and/or Serial (USB) connectivity.

It is strongly suggested to execute the routine immediately after the connection between central and peripherals has been established to minimize the effects of clock drift on the estimated clock offset, and therefore guarantee the best performance in terms of time synchronization.

2.2 Procedure

The time sync procedure is divided into 2 building blocks:

1. Connection (Figure 1), which cover device enumeration and connection as well as the system status check and settings of the current datetime.

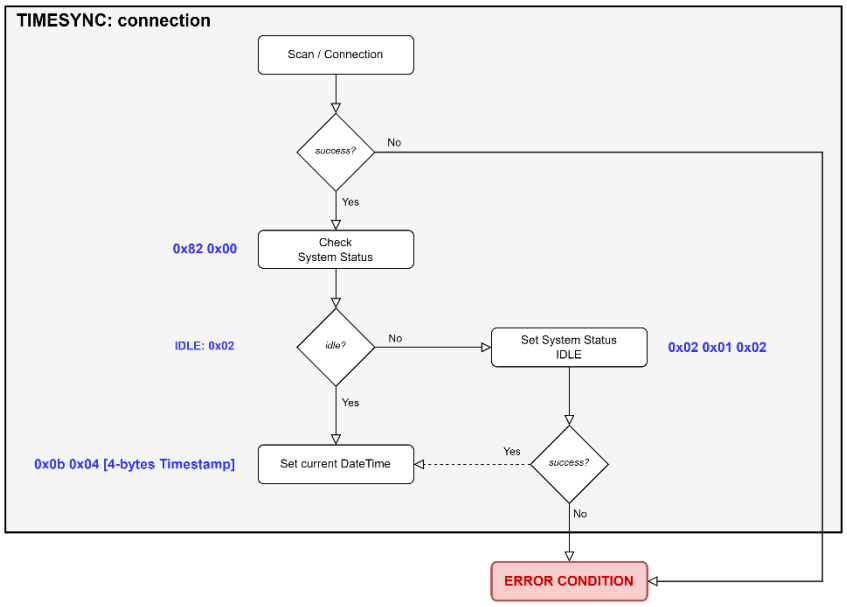

FIGURE 1: TIMESYNC CONNECTION STEP.

As regards the device enumeration and connection, the central node must subscribe to notification on the following command and data characteristics made available through the **Custom Service** with UUID **c8c0a708-e361-4b5e-a365-98fa6b0a836f**:

Command Characteristic

UUID:	d5913036-2d8a-41ee-85b9-4e361aa5c8a7
Byte:	20
Property:	READ, WRITE, NOTIFY
Description:	It allows to control the device behaviors.

Data Characteristic

UUID:	09bf2c52-d1d9-c0b7-4145-475964544307
Byte:	128
Property:	READ, NOTIFY, WRITE_NO_RESPONSE
Description:	It allows to manage streaming of data from device.

Once the connection is established, the device status must be checked using the following command:

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	82	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	82	ERROR CODE	SYSTEM STATE	0 … 0														

SYSTEM STATE[1]: it provides the HEX code corresponding to the current state of the system, which must be IDLE (i.e., 0x02).

If the device status is consistent, the next step is to set the current date and time by using the following command:

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	0b	04	TIMESTAMP TO BE SET				0 … 0													

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	0b	ERROR CODE	0 … 0															

TIMESTAMP TO BE SET[1]: it is the 32-bit unsinged integer value that represents the timestamp to be set, in Unix epoch format. Example (Figure 2):

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	0b	04	00	fa	bf	63	0 … 0													

“00 fa bf 63” which corresponds to “1/12/2023 12:16:00 PM”.

```
// Set datetime before starting time sync procedure
TimeSpan timeSpan = DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc));
uint secondsSinceEpoch = (uint)timeSpan.TotalSeconds;

byte[] argbytes = BitConverter.GetBytes(secondsSinceEpoch);

byte[] data = new byte[6];
data[0] = 0x0b;
data[1] = 0x04;
Array.Copy(argbytes, 0, data, 2, 4);

// Write command to command characteristic using the current connected device object
await dev.SelectedCmdCharacteristic.WriteAsync(data);
```

FIGURE 2: CODE SNIPPET EXAMPLE OF DATE SETTING.

This is of paramount importance to ensure the correct execution of subsequent operations.

[1] For further details about system states and commands necessary to manage get and set operations, please refer to the communication protocol description available here.

2. Execution (Figure 3Figure 1), which includes 3 phases.

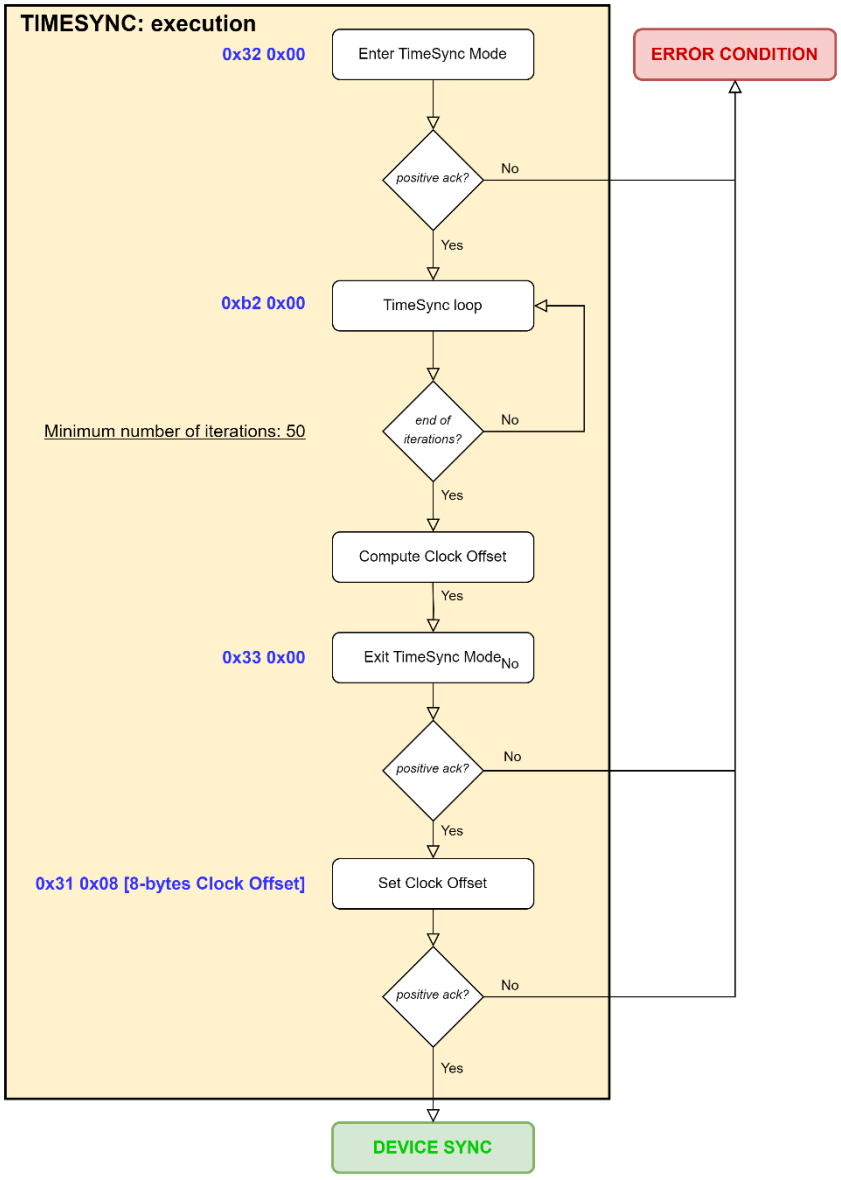

FIGURE 3: TIMESYNC EXECUTION STEP.

a. Entering timesync mode (Figure 4).

TRANSMISSION

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TYPE	**LENGTH**	**VALUE (Payload)**																	
32	00	0 … 0																	

RESPONSE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TYPE	**LENGTH**	**VALUE (Payload)**																	
00	02	32	ERROR CODE	0 … 0															

```
// Enter time sync mode
data = new byte[2] { 0x32, 0x00 };
await dev.SelectedCmdCharacteristic.WriteAsync(data);
```

FIGURE 4: CODE SNIPPET EXAMPLE OF ENTER TIMESYNC COMMAND.

b. Execution of a timesync loop (Figure 5), necessary to estimate the clock offset of that specific central / peripheral pair of devices. The loop starts with the acknowledge to enter time sync command (left side of Figure 5) and revolves around iteratively requesting a certain number of timestamps (right side of Figure 5). It is strongly recommended to execute at least 50 iterations.

Every time a get timesync command is written, the device will notify a 64-bit unsigned integer value representing the current timestamp in epoch format, with milliseconds resolution.

TRANSMISSION

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TYPE	**LENGTH**	**VALUE (Payload)**																	
b2	00	0 … 0																	

RESPONSE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TYPE	**LENGTH**	**VALUE (Payload)**																	
00	02	b2	ERROR CODE	TIMESTAMP									0 … 0						

```
// CMD_ENTER_TIME_SYNC
if (messageValue[2] == 0x32)
{
    if (messageValue[3] == 0)
    {
        // POSITIVE ACK
        _timesyncDriftItCounter = 50;
        if (_timesyncDriftItCounter > 0)
        {
            // Init timesync variables
            _clockDrift = 1;
            _result = 0;
            _nodeList = new List<UInt64>();
            _centralList = new List<double>();

            // Get current timestamp on master (e.g., PC) - first point of measure
            _T1 = HighResolutionDateTime.UtcNow;

            // Get current timestamp on slave (e.g., Muse)
            byte[] data = new byte[2] { 0xb2, 0x00 };
            await SelectedCmdCharacteristic.WriteAsync(data);

            _timesyncDriftItCounter--;
        }
        else
        {
            // Exit Timesync mode
            byte[] data = new byte[2] { 0x33, 0x00 };
            await SelectedCmdCharacteristic.WriteAsync(data);
        }
    }
    else
    {
        // NEGATIVE ACK: exit timesync mode
        byte[] data = new byte[2] { 0x33, 0x00 };
        await SelectedCmdCharacteristic.WriteAsync(data);
    }
}
```

```
// CMD_GET_TIME_SYNC
if (messageValue[2] == 0xb2)
{
    if (messageValue[3] == 0)
    {
        // POSITIVE ACK
        if (_timesyncDriftItCounter > 0)
        {
            // Get current timestamp on master (e.g., PC) - second point of measure
            _T4 = HighResolutionDateTime.UtcNow;

            // Decode the timestamp received from device
            byte[] buffer = new byte[8];
            Array.Copy(messageValue, 4, buffer, 0, 8);
            UInt64 timestamp = BitConverter.ToUInt64(buffer, 0);

            // Compute clock offset iteratively
            _result += _T4.Subtract(new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)).TotalMilliseconds;
            _result += _T1.Subtract(new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)).TotalMilliseconds;
            _result -= (ulong)1580000000 * 1000 * 2;    // REFERENCE EPOCH OF MUSE DEVICE
            _result -= timestamp;

            // Get current timestamp on master (e.g., PC) - first point of measure
            _T1 = HighResolutionDateTime.UtcNow;

            // Get current timestamp on slave (e.g., Muse)
            byte[] data = new byte[2] { 0xb2, 0x00 };
            await SelectedCmdCharacteristic.WriteAsync(data);

            _timesyncDriftItCounter--;
        }
        else
        {
            // Complete clock offset computation
            _result /= 50;
            _result /= 2;

            // Clock offset estimated, exit Timesync mode
            byte[] data = new byte[2] { 0x33, 0x00 };
            await SelectedCmdCharacteristic.WriteAsync(data);
        }
    }
    else
    {
        // NEGATIVE ACK: exit timesync mode
        byte[] data = new byte[2] { 0x33, 0x00 };
        await SelectedCmdCharacteristic.WriteAsync(data);
    }
}
```

FIGURE 5: CODE SNIPPET EXAMPLE OF TIMESYNC LOOP, WHICH START WITH THE ACKNOWLEDGE ON ENTERING TIME SYNC COMMAND.

c. Exit timesync mode and set clock offset.

To terminate the procedure, the exit command must be written (Figure 6).

```
// Exit Timesync mode
byte[] data = new byte[2] { 0x33, 0x00 };
await SelectedCmdCharacteristic.WriteAsync(data);
```

FIGURE 6: CODE SNIPPET EXAMPLE OF EXIT TIME SYNC COMMAND.

Once the central node receive the acknowledge to the exit time sync command, the last estimated clock offset value can be set (Figure 7).

```
// CMD_EXIT_TIME_SYNC
if (messageValue[2] == 0x33)
{
    if (messageValue[3] == 0)
    {
        // POSITIVE ACK

        // Build set clock offset message
        byte[] data = new byte[10];
        data[0] = 0x31;
        data[1] = 0x08;

        if (Math.Abs(_result) > 0)
        {
            byte[] clockOffset_bts = BitConverter.GetBytes((UInt64)Math.Abs(_result));
            Array.Copy(clockOffset_bts, 0, data, 2, 8);
        }

        // Write current estimated clock offset to command characteristic
        await SelectedCmdCharacteristic.WriteAsync(data);
    }
}
```

FIGURE 7: CODE SNIPPET EXAMPLE OF SET CLOCK OFFSET COMMAND.