```
/*

   _____                                                _____
__  _       __
  / ___/___  ____  ____  ___  ____ ___  ____ / ___/___
____  _____/ /_(_)____/ /__
  \__ \/ _ \/ __ \/_  / / _ \/ __ `__ \/ __ \\__ \/ _ \/
__ \/ ___/ __/ / ___/ //_/
 ___/ /  __/ / / / / /_/  __/ / / / / / /_/ /__/ /
__/ / / (__  ) /_/ / /__/ ,<
/____/\___/_/ /_/ /___/\___/_/ /_/ /_/\____/____/\___/
_/ /_/____/\__/_/\___/_/|_|

  SAM20 HW 2.0 - FW 1.0
*/


function decodeUplink(input)
{
    const bytes = input.bytes || [];
    const port = input.fPort;

    function readInt16BE(data, index)
    {
        var value = (data[index] << 8) | data[index + 1];

        if (value & 0x8000)
        {
            value -= 0x10000;
        }

        return value;
    }

    function readUint16BE(data, index)
    {
        return (data[index] << 8) | data[index + 1];
    }

    function readUint24BE(data, index)
    {
        return (data[index] * 0x10000) |
               (data[index + 1] << 8) |
                data[index + 2];
```

```
}

function result(data)
{
    return {
        data: data,
        warnings: [],
        errors: []
    };
}

function error(message)
{
    return {
        data: {},
        warnings: [],
        errors: [message]
    };
}

// Alarm packet - port 1
if (port == 1)
{
    return result({
        Status: bytes[0]
    });
}

// Data packet - port 2
if (port == 2)
{
    // Data packet without status
    if (bytes.length == 8)
    {
        return result({
            Status: 0,
            Temperature: readInt16BE(bytes, 0) / 100,
            Humidity: readUint16BE(bytes, 2) / 100,
            AirPressure: readUint16BE(bytes, 4) / 10,
            BatteryLevel: readUint16BE(bytes, 6)
        });
    }

    // Data packet with status
```

```
        if (bytes.length == 9)
        {
            return result({
                Status: bytes[0],
                Temperature: readInt16BE(bytes, 1) / 100,
                Humidity: readUint16BE(bytes, 3) / 100,
                AirPressure: readUint16BE(bytes, 5) / 10,
                BatteryLevel: readUint16BE(bytes, 7)
            });
        }

        // Data packet without status, with logging
record
        if (bytes.length == 20)
        {
            return result({
                Status: 0,
                Temperature: readInt16BE(bytes, 0) / 100,
                Humidity: readUint16BE(bytes, 2) / 100,
                AirPressure: readUint16BE(bytes, 4) / 10,
                BatteryLevel: readUint16BE(bytes, 6),
                Logging: {
                    FCnt: readUint24BE(bytes, 8),
                    Status: bytes[11],
                    Temperature: readInt16BE(bytes, 12) /
100,
                    Humidity: readUint16BE(bytes, 14) /
100,
                    AirPressure: readUint16BE(bytes,
16) / 10,
                    BatteryLevel: readUint16BE(bytes, 18)
                }
            });
        }

        // Data packet with status and logging record
        if (bytes.length == 21)
        {
            return result({
                Status: bytes[0],
                Temperature: readInt16BE(bytes, 1) / 100,
                Humidity: readUint16BE(bytes, 3) / 100,
                AirPressure: readUint16BE(bytes, 5) / 10,
                BatteryLevel: readUint16BE(bytes, 7),
```

```
                Logging: {
                    FCnt: readUint24BE(bytes, 9),
                    Status: bytes[12],
                    Temperature: readInt16BE(bytes, 13) /
100,
                    Humidity: readUint16BE(bytes, 15) /
100,
                    AirPressure: readUint16BE(bytes,
17) / 10,
                    BatteryLevel: readUint16BE(bytes, 19)
                }
            });
        }
    }

    // Config packet - port 3
    if (port == 3)
    {
        const DataRatePlusADR = bytes[4];

        return result({
            Status: bytes[0],
            SendPeriod: bytes[1],
            MoveThr: bytes[2],
            PacketConfirm: bytes[3],
            DataRate: DataRatePlusADR & 0x0F,
            ADRon: Boolean(DataRatePlusADR & 0x80),
            FamilyId: bytes[5],
            ProductId: bytes[6],
            HW: bytes[7] / 10,
            FW: bytes[8] / 10
        });
    }
}
```