Binary Protocol 8-bit (harp-1.0)

Table of Contents:

Introduction

The Harp Protocol is a binary communication protocol created in order to facilitate and unify the interaction between different devices. It was designed with efficiency and ease of parsing in mind.

The protocol is based on addresses. Each address points to a certain memory position available in the device. These positions are called registers. Each register is defined by a data type and some meaningful functionality attached to the data.

The Harp Binary Protocol is commonly used for all exchanges between a Controller and a Device. The Controller can be a computer, or a server and the Device can be a data acquisition or actuator microcontroller.

The available packets are:

- Command: Sent by the Controller to the Device. Command messages can be used to read or write the register contents.

- Reply: Sent by the Device in response to a Command.
- Event: Sent by the Device when an external or internal event of interest happens. An Event message will always carry the contents of the register that the event refers to.

> **Note**
>
> The Harp Binary Protocol uses Little-Endian byte ordering.

Harp Message specification

The Harp Message contains a minimal amount of information to execute a well-defined exchange of data. It follows the structure below.

== Harp Message == [`MessageType`] [`Length`] [`Address`] [`Port`] [`PayloadType`] [`Payload`] [`Checksum`]

[`MessageType`] (1 byte)

Specifies the type of the Harp Message.

Value	Description
1 (Read)	Read the content of the register with address [RegisterAddress]
2 (Write)	Write the content to the register with address [RegisterAddress]
3 (Event)	Send the content of the register with address [RegisterAddress]

[`Length`] (1 byte)

Contains the number of bytes that are still available and need to be read to complete the Harp message (i.e. number of bytes after the field [`Length`]).

[`Address`] (1 byte)

Contains the address of the register to which the Harp Message refers to.

[`Port`] (1 byte)

If the device is a Hub of Harp Devices, it indicates the origin or destination of the Harp Message. If the field is not used or it's equal to `0xFFFFFFFF`, it points to the device itself.

[`PayloadType`] (1 byte)

Indicates the type of data available on the [Payload]. The structure of this byte follows the following specification:

7	6	5	4	3	2	1	0
IsSigned	IsFloat	0	HasTimestamp	Type			

Type (4 bits)

Specifies the size of the word in the [Payload].

Value	Description
1	8 bits
2	16 bits
4	32 bits
8	64 bits

HasTimestamp (1 bit)

If this bit is set the Harp Message contains a timestamp. In this case the fields [Seconds] and [Nanoseconds] must be present in the message.

IsFloat (1 bit)

This bit indicates whether the [Payload] represents fractional values. If the bit is not set, the payload contains integers.

IsSigned (1 bit)

If the bit is set, indicates that the [Payload] contains integers with signal.

> **Note**
>
> The bits [IsFloat] and [IsSigned] must never be set simultaneously.

[`Payload`] (? bytes)

The content of the Harp Message.

If [`PayloadType`] `HasTimestamp` flag is set, the following optional fields are present:

Seconds (4 bytes)

Contains the number of seconds (`U32`) of the Harp Timestamp clock. This field is optional. In order to indicate that this field is available, the bit [HasTimestamp] in the field [PayloadType] needs to be set.

Nanoseconds (2 bytes)

It contains the fractional part of the Harp Timestamp clock in nanoseconds (`U16` containing the number of microseconds divided by 32).

This field is optional. In order to indicate that this field is available, the bit [HasTimestamp] in the field [PayloadType] needs to be set.

> **Note**

> The full timestamp information can thus be retrieved using the formula: Timestamp(s) = [Seconds] + [Nanoseconds] * 32 * 10-6

[`Checksum`] (1 byte)

The sum of all bytes (`U8`) contained in the Harp Message. The receiver of the message should calculate the checksum and compare it with the received. If they don’t match, the Harp Message should be discarded.

Features and Code Examples

Some of the fields described on the previous chapter have special features. These are presented next.

[`MessageType`] and `Error Flag`

The field [Command] has an Error flag on the 4th least significant bit. When this bit is set it means that an error has occured. Examples of possible errors cane be:

1. The host tries to read from a register that doesn’t exist;
2. The host tries to write invalid data to a certain register;
3. The [PayloadType] doesn’t match the target register's `PayloadType`.

A simple code in C to check for error will be:

```
int errorMask = 0x08;

if (Command & errorMask)
{
printf(“Error detected.\n”);
}
```

Harp Message [Length]

If one byte is not enough to express the length of the Harp Message, use [Length] equal to 255 and add after an unsigned 16 bits word with the Harp Message length. Replace the [Length] with: [255] (1 byte) [ExtendedLength] (2 bytes)