April 2009

Henne's Sites

AN117: Remote Device Management [Responder]

Content

Preface

RDM is an extension of the DMX512 standard. In 2006 it was specified in ANSI E1.20 by the ESTA.
Using RDM, DMX-Receivers like dimmers or moving lights are able to respond to the DMX-Controller. Typically RDM is used for:

- changing the start addresses of Fixtures (Patch)
- changing the operating mode of Fixtures (Personality)
- Parameterization of Devices
- Retrieving status information (Errors, Temperatures, Currents,…)
- Firmware updates (in future)

RDM is backward compatible to DMX512-A. So older DMX Fixtures and new RDM Devices can be mixed in the same Bus.

Despite these great benefits you should think carefully about implementing an RDM protocol handler in your own devices:

- Currently there are just few expensive commercial RDM controllers.
- Most DMX-Splitters are unidirectional and won't support talk back.
- Each RDM Device needs a GUID.
- To respond sensor data you need sensors ;-)
- RDM is much more sensitive to termination and wiring errors than DMX.
- The frame format for firmware updates is not yet standardized.
- Despite all tweaks – With 2kB flash and 55 byte SRAM, the RDM responder lib needs much more memory than the DMX reception lib.

Terms of Use
You can use the code under the terms of the gnu general puplic license (GPL). If this causes problems, please contact the author.

Basics
In contrast to DMX, RDM is based on packets. The following table shows the structure of an RDM packet:

Slot No.	Function
0	alternate start code (0xCC)
1	sub start code (0x01)
2	packet length in bytes
3 - 8	target address (GUID)
9 - 14	source address (GUID)
15	transaction number
16	Port ID / message type
17	amount of waiting messages
18 - 19	sub device
20	command type
21 - 22	parameter ID
23	length of parameter data in bytes
24 – (n-2)	parameter data
(n-1) - n	check sum

Numbers are transmitted big-endian (most significant byte first).
The packet length is equal to the slot number of the first byte of the check sum.
The GUIDs are composed of the 16bit Manufacturer-ID, given by the ESTA, and a 32bit device ID.
The 16bit check sum is the sum of all bytes of the packet (not including the check sum itself).

Before the controller can send messages to specific responders, it has to discover the RDM-Responders connected to the bus. For this, the controller broadcasts (target GUID is 0xFFFFFFFFFFFF) a discovery message with the upper and lower limits of the current address space as parameter data. All Responders in that address space respond with their encrypted GUID. All found devices are muted then.
The ESTA proposes a tree search as discovery algorithm.

For detailed information please read the E1.20.

Responder Library

The code was written for the DMX-Transceiver Rev. 3.01 but should be portable to the most AVRs of the mega series. The main objective of this library was the support of the most important RDM features with still a small code size. The code was written with AVR Studio 4.13 and WinAVR-20060125.

With „init_RDM()“ the RDM and DMX reception is initialized.
“check_rdm()” handles received RDM packets and should be called as often as possible in the main task.

UID_0 and UID_1 are the ESTA Manufacturer-ID.
UID_2 through UID_5 are the Device-ID.

F_OSC is the frequency of the crystal in kHz.

When new DMX data is received the Flag „EVAL_DMX“ in „Flags“ is set. The data is located in “DmxField[]”.

If „DEBUG“ is defined, LEDs connected to PortA indicate the following states:

- PA0 changes its state if an RDM packet with a valid start code arrives.
- PA1 changes its state if the packet fits in the RDM buffer.
- PA2 changes its state if the check sum is valid.
- PA3 changes its state if a Discovery Request is answered.
- PA4 is high if the device is muted.

Currently not implemented features:

- Sub-Devices
- Queued Messages
- Acknowledge Timer
- some parameters

Description of the code
If the start code after the Break is 0xCC and the following byte is 0x01, we expect an RDM packet. If the message buffer is empty (the last message is handled) the message will be received.
The third byte is the packet length. If the buffer is big enough for the message, we receive the given number of bytes, buffer them and build the check sum in “RxCount_16”.
After all data is received we compare the transmitted check some with the calculated one and set the flag “EVAL_RDM” if they are equal.

The next time, „check_rdm()“ is called, the message is handled if we are the receiver of it and the parameter is supported.

To reduce code size, predefined response data is read directly from flash.

Finally the response packet is built and transmitted by calling “respondMsg()”.

If an RDM packet is received, the green LED changes its state.
If a response is transmitted, the red LED changes its state.

Annotations
If you fixed bugs or improved the library, I will be glad to hear from you.
Extensions of the lib are welcome, too ;-)