April 2009

Henne's Sites

AN112: DMX-Reception with AVRs

Introduction
In this application note the reception of DMX signals with AVR controllers is described. There is an Assembler- and a C-Version of the used state machine. This application note was written for the DMX-Transceiver but should be portable to most AVRs.

Thanks to Jörgen Dierking for his support.

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

DMX 512
DMX512 is a unidirectional differential serial protocol based on the physical layer of RS485. The baudrate is 250kbit/sec. There are one master and up to 32 slaves in one bus. The number of slaves can be increased by using splitters/boosters. One universe contains up to 512 channels. The transmission is initialized by a break (LO level) of >88µs, a mark after break (MAB, HI level) of >8µs and a start byte (also called start code) which is usually 0. A byte is made of 1start bit, 8data bits and 2stop bits (8n2).

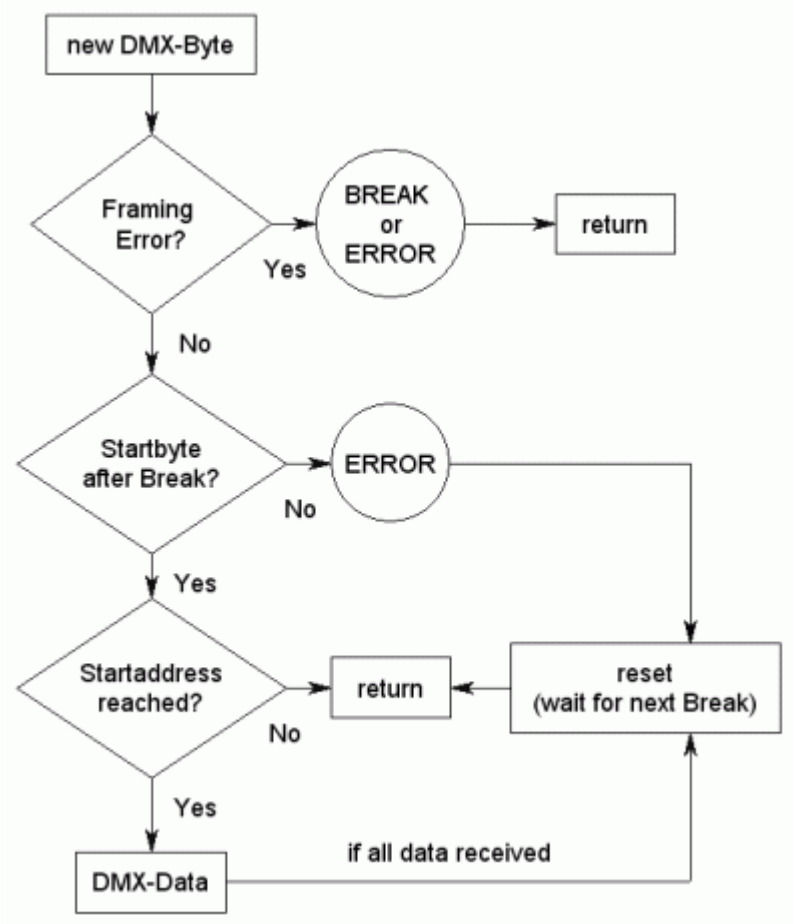
Description of the code
A reception without errors looks like this:

After getting the data byte and the state from the USART, we recognize a framing error, which could be a break. Believing it is a break, we store that in the DMX state and return.

If there was a break before, the next byte must be the start byte (=0). If the byte is really 0, we use the X register as a frame counter and set it to the start address. If this byte is not 0, there was a real framing error and we wait for the next break without processing further frames.

Now the frame counter is decremented with each frame untill the start address is reached. After that the DMX state is incremented again.

The next frames are the channel data you want to get. If all desired channels are received, the DMX state is cleared and we wait for the next break.



Assembler-Version
The code was written with AVR Studio 4.13

```
#define DMX_FIELD       0x60        //base address of DMX array
#define DMX_CHANNELS    2           //no. of channels
#define DMXstate        R19         //status reg for DMX-ISR (r16-r31)

#define F_OSC           8000        //oscillator freq. in kHz (typical 8MHz or 16MHz)
#define USE_DIP                     //should get start address from DIPs?
```

DMX_FIELD is the first channel in the receive buffer located in the SRAM. The SRAM starts with 0x60.
DMX_CHANNELS is the amount of desired DMX channels of the device.
DMXstate is used as state register in the ISR.
F_OSC is the CPU frequency in kHz.
When USE_DIP is defined the DIP switches of the transceiver are used for getting the start address.

The USART is initialized for DMX reception and the receive buffer is cleared by calling “init_dmx”.

With “get_byte” as USART RX-Complete ISR, DMX data is received.

You have random access to the DMX data in the DMX_FIELD.

The library should be included at the end of the entry file (but still before tables…).

C-Version

The code was written with AVR Studio 4.13 and WinAVR-20060125.
In the header file the following constants and varables have to be defined:

```
#define  F_OSC   8000                    //oscillator freq. in kHz (typical 8MHz or 16MHz)
#define  USE_DIP                         //get start address from DIPs

volatile  uint8_t    DmxRxField[8];      //array of DMX vals (raw)
volatile  uint16_t   DmxAddress;         //start address

extern void          init_DMX_RX(void);
extern void          get_dips(void);
```

F_OSC is the CPU frequency in kHz.

When USE_DIP is defined, the DIP switches of the transceiver are used for getting the start address. Otherwise the start address can be written directly to DmxAddress.

The desired DMX data is stored in DmxRxField[].

The USART is initialized for DMX reception and the receive buffer is cleared by calling “init_DMX_RX()”.

When calling “get_dips()” the start address is calculated from the DIP switches.