USB IR Remote Receiver (AVR Multiprotocol Receiver) description

General description of the device

German Project site: USB IR Remote Receiver

The USB IR Remote Receiver is based on Frank M. IRMP - Infrarot-Multiprotokoll-Decoder and V-USB.
The main motivation for this project was the outdated Igor Plug remote receiver. As Frank M. is descripting RC5 is obsolete I decided to combine Frank M. multiprotocol receiver with V-USB to be able to receive also other protocols through the USB port. Also newer motherboards do not always include a serial COM port.

The device is working as low speed USB 1.1 device. As it get recognized as HID device on Windows no special driver is needed (tested on Windows 7 x86 and WinXP SP3).

The firmware can be easily updated if the modified bootloadHID get programed first on the device.

The USB IR Remote Receiver has also a PowerOn function to start/shutdown the host PC through the remote control. The first received IR code will be stored in EEPROM.
If a received IR code equals the stored code the device will switch on a output pin for ~250ms. (see AVR source description).
With the modified bootloaderHID it is possible to update the firmware of the device without needing to set a jumper or to disconnect/reconnect the device.

Supported software:
DVBViewer: copy 'USB_IR_Remote_Receiver.dll' to \DVBViewer\Plugins and enable the input plugin in the DVBViewer options
Girder: copy 'USB_IR_Remote_Receiver.dll' to \Girder\Plugins and enable the plugin in the Girder options. Girder versions since 3.2.9 are supported
EventGhost: copy 'USB_IR_Remote_Receiver.dll' and '__init__.py' to \EventGhost\plugins\USBIRRemoteReceiver and add the plugin in EventGhost
Native: also the 'USB_IR_Remote_Receiver.dll' can be used for self made programs with the functions InitNative and InitPAnsiChar (see below)

Right now these IR protocols are supported by the included IRMP version from '26.06.2010':

Protokoll	Hersteller
SIRCS	Sony
NEC	NEC, Yamaha, Canon, Tevion, Harman/Kardon, Hitachi, JVC, Pioneer, Toshiba, Xoro, Orion, NoName und viele weitere japanische Hersteller.
SAMSUNG	Samsung
SAMSUNG32	Samsung
MATSUSHITA	Matsushita
KASEIKYO	Panasonic, Technics, Denon und andere japanische Hersteller, welche Mitglied der "Japan's Association for Electric Home Application" sind.
RECS80	Philips, Nokia, Thomson, Nordmende, Telefunken, Saba
RECS80EXT	Philips, Technisat, Thomson, Nordmende, Telefunken, Saba
RC5	Philips und andere europäische Hersteller
DENON	Denon
RC6	Philips und andere europäische Hersteller
APPLE	Apple
NUBERT	Nubert, z.B. Subwoofer System
B&O	Bang & Olufsen
GRUNDIG	Grundig
NOKIA	(Nokia) D-Box 2
SIEMENS	Siemens, z.B. Gigaset M740AV (ab ~15kHz, siehe Bemerkungen zu F_INTERRUPTS)
FDC	FDC Keyboard (ab ~15kHz)
RCCAR	RC Car: IR Fernbedienung für Modellfahrzeuge

Change log

28.06.2010:
DLL: v1.0.0.9, added new protocols to standard list
AVR: Updated Irmp to 26.06.2010, new protocol FDC and RCCAR
Because of the limitation by the memory of the Atmega8 it is not anymore possible to enable all protocols in the 'irmpconfig.h'. Just recompile the project for your needs.

06.06.2010:
DLL: v1.0.0.8, added new protocols to standard list,
New: If a new protocol is missing in the INI file it will be added automatically
AVR: Updated Irmp to 02.06.2010, new protocol NOKIA and SIEMENS, bugfix Grundig

18.05.2010:
DLL: v1.0.0.7, just added new protocol to standard list,
AVR: Updated Irmp to 17.05.2010, new protocol GRUNDIG, bugfix SAMSUNG32

09.05.2010:
DLL: v1.0.0.6, Fixd bug with standby and disconnecting the device
EventGhost: Added plugin to be able to use the device with EventGhost

02.05.2010:
Docu: Updated schematic for using PonyProg2000 with the LPT port. Added PonyProg2000 flashing description.

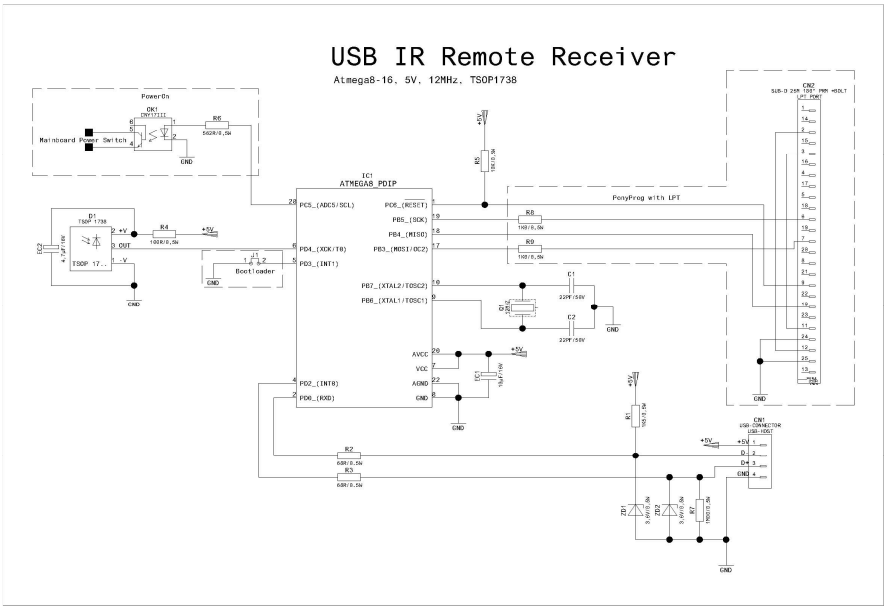
28.04.2010:
DLL: v1.0.0.5, Added possibility to flash or update the device firmware direct with the USB IR Remote Receiver settings/option dialog.
AVR: Bugfix code repeats,
Updated Irmp to 28.04.2010,
Added modified booloadHID to be able to flash the device without setting a jumper or needed disconnect/reconnect of the device
The booloadHID will boot if:
+ no USB IR Remote Receiver firmware is flashed; + jumper is set
+ the firmware itself is reseting the device for bootloader modus

23.04.2010:
DLL: v1.0.0.4, Added Key Suppression, Bugfix of DVBViewer on showing settings/option dialog
AVR: optimized detection for PowerON IR code, Bugfix for more than 255 IR code repeats

16.04.2010:
DLL: v1.0.0.3, Added InitNative, InitPAnsiChar and Girder functions.
AVR: Updated Irmp to 16.04.2010

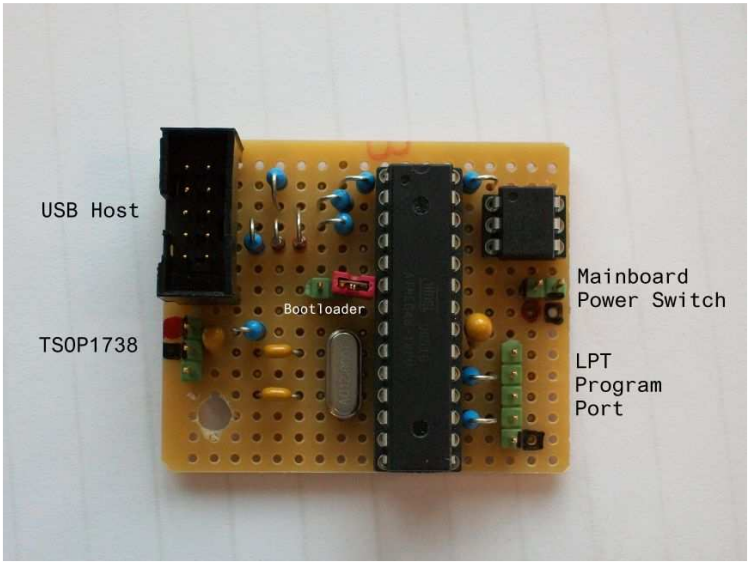
12.04.2010:
AVR: Updated Irmp to 12.04.2010, new protocol B&Q

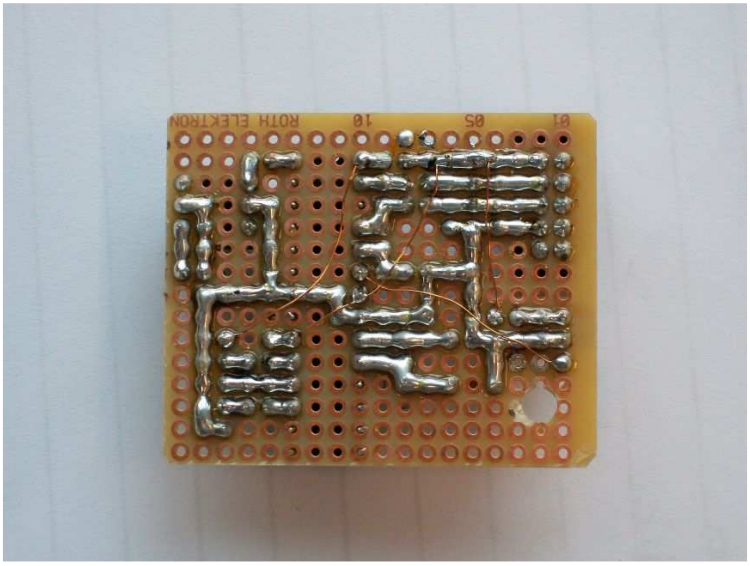
Hardware schematic



The diodes ZD1 and ZD2 are needed to comply to the USB standard.
The IR receiver can be also one of this types:
Vishay TSOP 1738, Vishay TSOP 1838, Vishay TSOP 11xx series, Siemens SFH 5110, Radio Shack 276-0137, Mitsumi IR Preamp KEY-COOSV (0924G), Toshiba TK19 444 TFMS 5360, Temic TFMS 5380 by Telefunken Semiconductors, Sharp IS1U60, Everlight IRM-8601S, Sony SBX 1620-12, Sharp GP1U271R
Optional: OK1 if the PowerOn function is used to start/shutdown the host PC
Optional: J1 if a bootloader will be used: bootloadHID
If the bootloader get be used be sure to write the right fuse bits. Otherwise the bootloader will be deleted after the first flashing.
With the included modified bootloadHID this jumper isn't needed anymore.
Optional: CN2, R8 and R9: Only needed once to download the modified bootloadHID with PonyProg2000.

Picture of an assembled USB IR Remote Receiver:





AVR Source description

To compile the source Atmel AVR Studio and WinAVR will be needed.

irmpconfig.h:
(pin can be changed if the IR receiver diode is not connected to PD4)

```
#define IRMP_PORT  PORTB
#define IRMP_DDR   DDRD
#define IRMP_PIN   PIND
#define IRMP_BIT   4       // use PD4 as IR input
```

main.c:

```
//enter here the Irmp build date:
const char IrmpVersion[] = "28.04.2010";
```

configUSBIRRemoteReceiver.h:

```
#define USE_BOOTLOADER          1                                   /* 1 if bootloader option should be included (default), 0 to disable */

#define USE_PowerOnFunction     1                               /* 1, use PowerOn function (default), 0 to disable */

#ifdef USE_PowerOnFunction
#define SWITCH_PORT             PORTC                               /* PORTx - register for Switch output */
#define SWITCH_BIT              PC5                                 /* bit where OK1 will be connected */
#define SWITCH_DDR              DDRC                                /* Switch data direction register */
#endif
```

irmpconfig.h:

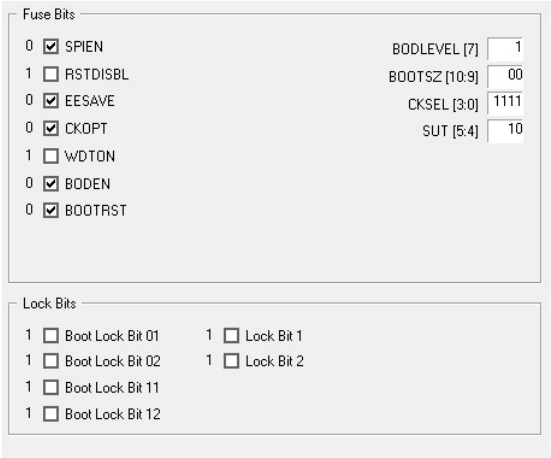
```
/*---------------------------------------------------------------------------------------------------------------------------------------------------
 * Change settings from 1 to 0 if you want to disable one or more decoders.
 * This saves program space.
 *
 * 1 enable  decoder
 * 0 disable decoder
 *
 * The standard decoders are enabled per default.
 * Some less common protocols are disabled here, you need to enable them manually.
 *
 * If you want to use FDC or RCCAR simultaneous with RC5 protocol, additional program space is required.
 * If you don't need RC5 when using FDC/RCCAR, you should disable RC5.
 *---------------------------------------------------------------------------------------------------------------------------------------------------
*/

//      Protocol                                Enable  Remarks                 F_INTERRUPTS        Program Space
#define IRMP_SUPPORT_SIRCS_PROTOCOL             1       // Sony SIRCS           >= 10000                ~100 bytes
#define IRMP_SUPPORT_NEC_PROTOCOL               1       // NEC + APPLE          >= 10000                ~250 bytes
#define IRMP_SUPPORT_SAMSUNG_PROTOCOL           1       // Samsung + Samsung32  >= 10000                ~250 bytes
#define IRMP_SUPPORT_MATSUSHITA_PROTOCOL        1       // Matsushita           >= 10000                 ~50 bytes
#define IRMP_SUPPORT_KASEIKYO_PROTOCOL          1       // support Kaseikyo     >= 10000                 ~50 bytes
#define IRMP_SUPPORT_DENON_PROTOCOL             1       // support DENON        >= 10000                ~250 bytes
#define IRMP_SUPPORT_RC5_PROTOCOL               1       // RC5                  >= 10000                ~250 bytes
#define IRMP_SUPPORT_RC6_PROTOCOL               0       // RC6                  >= 10000                ~200 bytes
#define IRMP_SUPPORT_GRUNDIG_PROTOCOL           1       // Grundig              >= 10000                ~150 bytes
#define IRMP_SUPPORT_NOKIA_PROTOCOL             1       // Nokia                >= 10000                ~150 bytes
#define IRMP_SUPPORT_NUBERT_PROTOCOL            1       // NUBERT               >= 10000                 ~50 bytes
#define IRMP_SUPPORT_BANG_OLUFSEN_PROTOCOL      0       // Bang & Olufsen       >= 10000                ~200 bytes
#define IRMP_SUPPORT_FDC_PROTOCOL               1       // FDC3402 keyboard     >= 10000 (better 15000)  ~50 bytes (~400 in combination with RC5)
#define IRMP_SUPPORT_RCCAR_PROTOCOL             0       // RC Car               >= 10000 (better 15000) ~150 bytes (~500 in combination with RC5)
#define IRMP_SUPPORT_SIEMENS_PROTOCOL           0       // Siemens Gigaset      >= 15000                ~150 bytes
#define IRMP_SUPPORT_RECS80_PROTOCOL            0       // RECS80               >= 20000                 ~50 bytes
#define IRMP_SUPPORT_RECS80EXT_PROTOCOL         0       // RECS80EXT            >= 20000                 ~50 bytes
```

bootloadHID

This packet is including a modfied version of bootloadHID. The new feature of the modifed bootloader is that no jumper is needed anymore.
If the modified bootloadHID got programed on the device the firmware can be updated by the USB_IR_Remote_Receiver.dll settings/option.
The bootloader needs 2048 bytes of boot flash.

If the bootloader is getting programmed be sure to set the right fuse bits!
(Take a look to the description below how to program the device with PonyProg2000)

If the bootloader position in the flash will be needed to be adjusted change the bootloader address in the AVR-Studio options:
Project -> Configuration Options -> Custom Options -> [Linker Options]
Edit the line *-Wl,--section-start=.text=0x1800* for your device. The bootloader address for the Atmega8 is 0x1800.




```
#############################################################################
# Configure the following variables according to your AVR. The example below
# is for an ATMega8. Program the device with
# make fuse  # to set the clock generator, boot section size etc.
# make flash # to load the boot loader into flash
# make lock  # to protect the boot loader from overwriting

DEVICE = atmega8
BOOTLOADER_ADDRESS = 1800
```

```
F_CPU = 12000000
FUSEH = 0xc0
FUSEL = 0x9f

# Fuse high byte:
# 0xc0 = 1 1 0 0   0 0 0 0 <-- BOOTRST (boot reset vector at 0x1800)
#        ^ ^ ^ ^   ^ ^ ^------ BOOTSZ0
#        | | | |   | +-------- BOOTSZ1
#        | | | |   + --------- EESAVE (preserve EEPROM over chip erase)
#        | | | +-------------- CKOPT (full output swing)
#        | | +---------------- SPIEN (allow serial programming)
#        | +------------------ WDTON (WDT not always on)
#        +-------------------- RSTDISBL (reset pin is enabled)
# Fuse low byte:
# 0x9f = 1 0 0 1   1 1 1 1
#        ^ ^ \ /   \--+--/
#        | |  |       +------- CKSEL 3..0 (external >8M crystal)
#        | |  +--------------- SUT 1..0 (crystal osc, BOD enabled)
#        | +------------------ BODEN (BrownOut Detector enabled)
#        +-------------------- BODLEVEL (2.7V)
################################################################################
```

Screen shot settings/option dialog

Device Connected: If the device got connected to the host the icon will change to green and the included Irmp build date will be shown. Also the USB IR Remote Receiver firmware will be shown.
PowerOn Function Status: If the PowerOn function is enabled the icon will be green. Use the button to enable/disable the PowerOn function. If the option "Disable PowerOn when initialized"
is checked the PowerOn function will be automatically disabled when the plugin get's initialized. So the IR code what is used for PowerOn can be used on the host for a different function.
Received IR Code: Will show the last received IR code
Trained IR Code: Will show the IR code what got stored in EEPROM for PowerOn
Manage Trained IR Code:
Read Trained IR Code: Reads the IR code stored in EEPROM
Clear Trained IR Code: Will clear the actual stored IR code in EEPROM. The next received IR code will be stored as new PowerOn IR code.
IR Polling Time: For different IR receiver diodes it is possible to adjust the polling time. A ussefull time is between 66 - 100µs
Minimum of Repeats: The device will debounce the remote control. With the minimum of repeats the number of needed repeats can be adjusted until the same IR code getting sent again to the host.
A repeat of 0 means no change to the remote control.
Flash/Update Firmware: Will show a new dialog to update the firmware of the device. This option is only available if firmware higher or equal 1.1 is used or the device is in booloader modus

Flash/Update firmware dialog

Load a valide Intel Hex file to flash/update the firmware of the device. The file size is limited by the device flash size minus 2048 bytes bootloader size.
If the USB IR Remote Receiver firmware is running the device will be reseted at first to bring the device in bootloader modus.
After the flashing the device will reboot again and start the new flashed firmware.

There is no need to set the jumper or to disconnect/reconnect the device if the modified bootloader is getting used.

USB_IR_Remote_Receiver.ini

[IRMP Protocols]
List of supported protocols of the device (depend on IRMP version).
If a new protocol is added to the device the DLL doesn't need to be updated. Just add the new protocol in this section:
name_of_protocol=IRMP_ID

[Settings]
Default the device is opened delayed with 2000ms. This value can be changed when adding:
StartUpDelay=desired_value_in_milliseconds

{Default 2000}

Default there is no key suppression. The received IR codes are getting forwared as they are getting decoded and received ba the DLL.
If the host software isn't fast enougth to handle fast repeating IR codes a timer for key suppresion can be used.
This option disallow a received IR to get sent to the host software if a previous received IR code got sent and the key suppression is still active.
Key Suppression=desired_value_in_milliseconds
{Default 0}

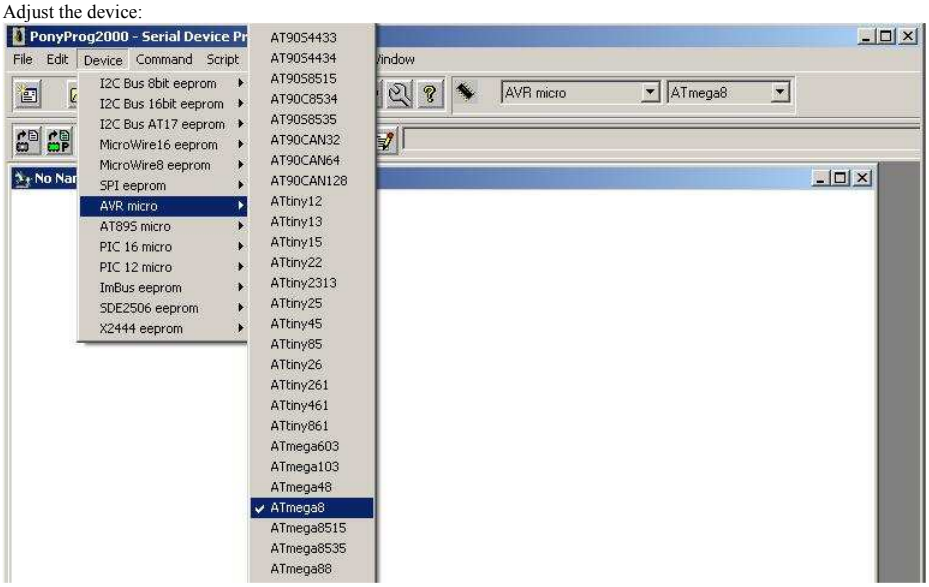
Add this value to enable a log memo in the settings dialog:
Debug=1
{Default 0}

Program the device with PonyProg2000

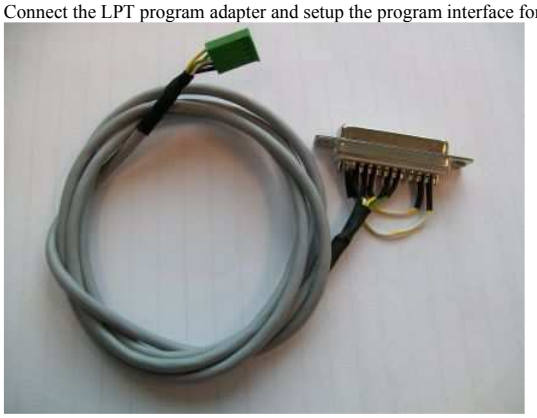
Download and install the latest beta version of PonyProg2000.

Run at first the calibration:

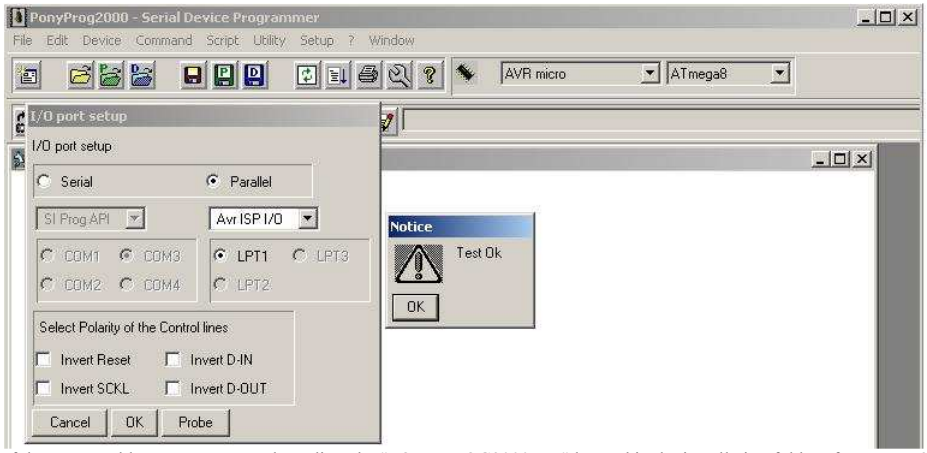
Adjust the device:



Connect the LPT program adapter and setup the program interface for LPT. Remember to connect also the USB-Host because the device will need +5V to operate:



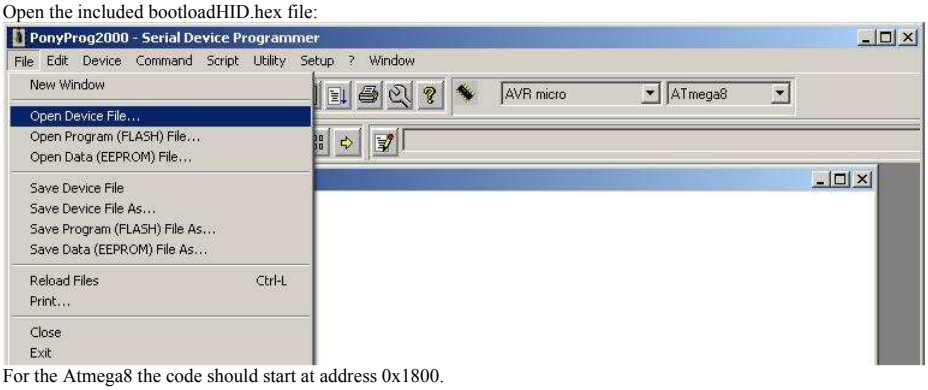
Push "Probe" to test the connection to the device:



If there are problems you may need to adjust the "PONYPROG2000.INI" located in the installation folder of PonyProg2000.

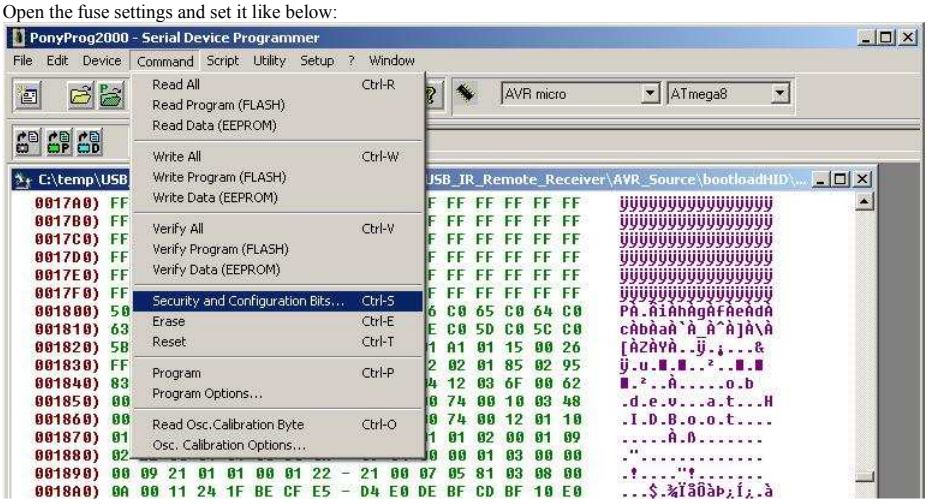
Set the value "AutoDetectPorts=YES" to NO and enter manually the LPT port address like "LPTPorts=3BC".
Take look to the device manager to find out your LPT address.

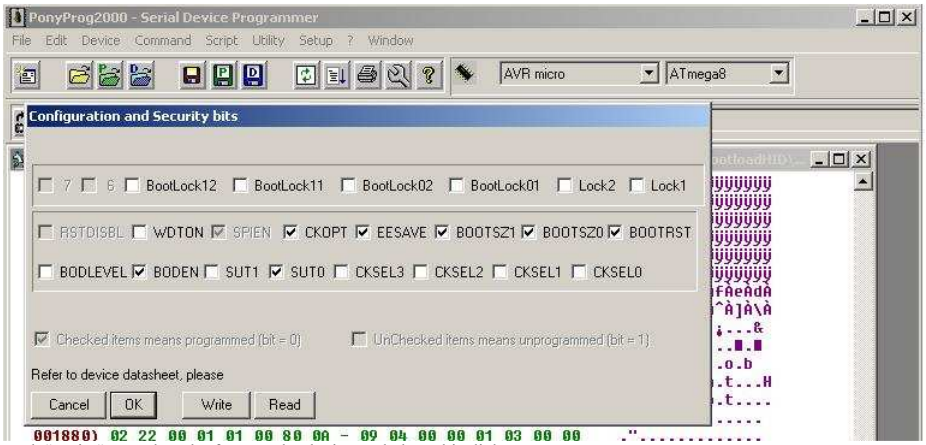
Open the included bootloadHID.hex file:



For the Atmega8 the code should start at address 0x1800.

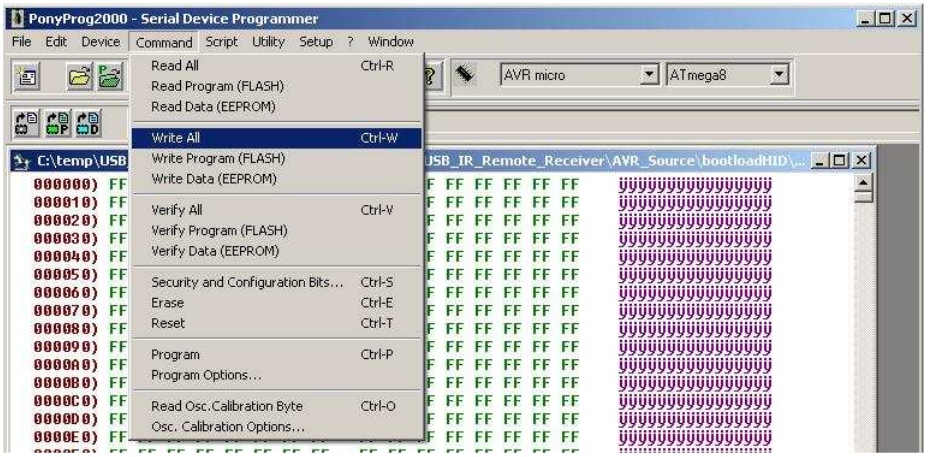
Open the fuse settings and set it like below:





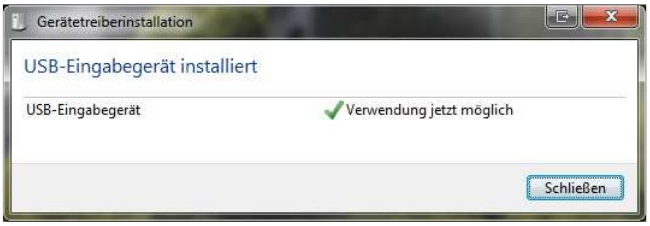
Push "Write" to update the fuses at the device and close this dialog.

After this use "Write All" to write the bootloadHID.hex on the device:



When it is finished disconnect the USB cable to power off the device. Disconnect also the LPT program adapter.
Plugin in again the USB cable and the device should be recognized as "BootHID" in Windows.
Now use the DLL settings/options to download a firmware.

USB IR Remote Receiver device installation on Windows 7 x86, just plug in the device. The driver will be installed automatically



USB IR Remote Receiver device installation on Windows XP SP3, just plug in the device. The driver will be installed automatically

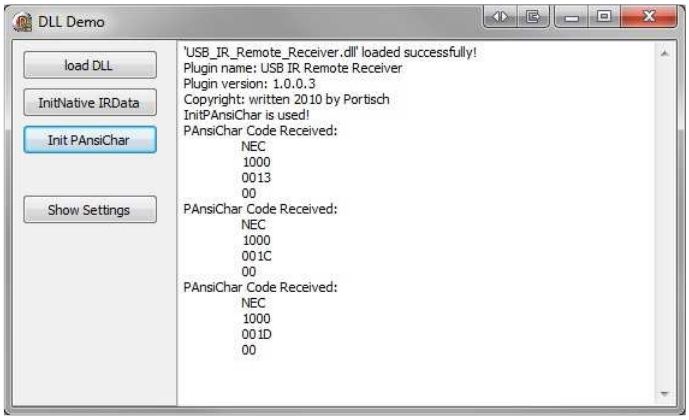
Function prototypes in "USB IR Remote Receiver.dll" library:

The DLL will handle the communication between the HID AVR device and the host.
The USB IR Remote Receiver DLL can be used directly with DVBViewer in the '\Plugins' directory as input plugin.
The USB IR Remote Receiver DLL can be also used directly with Girder in the '\Plugins' directory. Tested with Girder 3.2.9 (last Freeware version), 4.0.5.2 and 5.0.10.

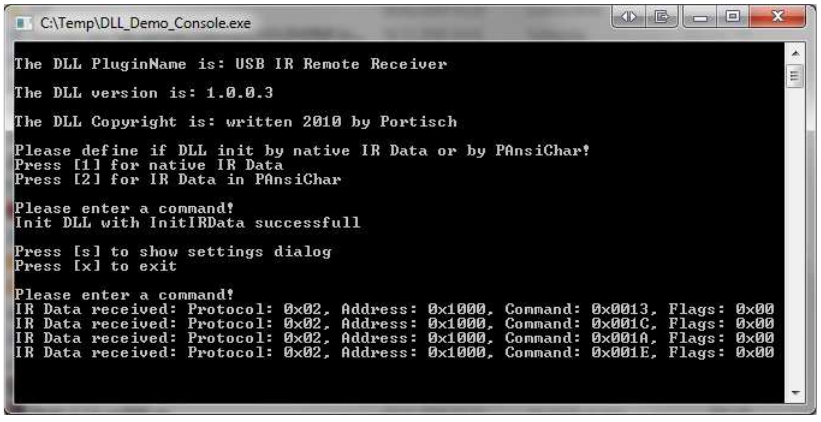
Also it can be used for other programs to handle the device with the functions InitNative or InitPAnsiChar.

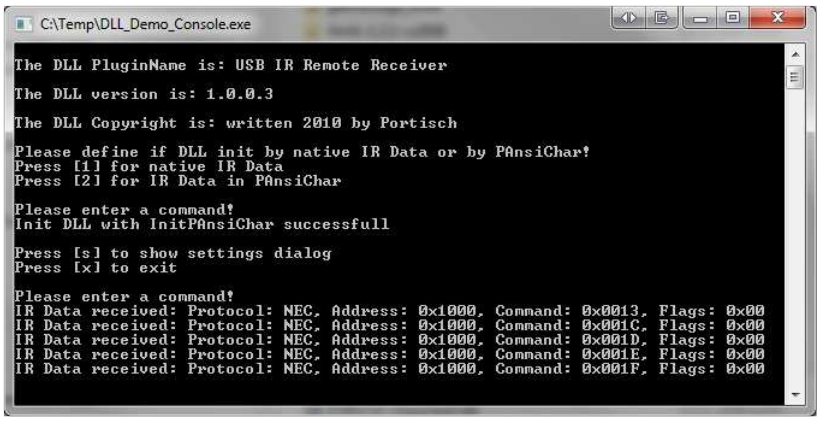
Screen shots from the demo sources

Example output with the DLL_Demo.exe (source Delphi 2010):



Example output with the DLL_Demo_Console.exe (source Visual Studio 2008):





Delphi:

```
function Version():PAnsiChar; stdcall external USBIRRemoteReceiver name 'Version';
function Copyright():PAnsiChar; stdcall external USBIRRemoteReceiver name 'Copyright';
function PluginName():PAnsiChar; stdcall external USBIRRemoteReceiver name 'PluginName';
procedure ShowSettings(Handle : THandle); stdcall external USBIRRemoteReceiver name 'ShowSettings';

function InitNative(Callback : TCallbackNativeIRData):boolean; stdcall external USBIRRemoteReceiver name 'InitNative';
function InitPAnsiChar(Callback : TCallbackPAnsiChar):boolean; stdcall external USBIRRemoteReceiver name 'InitPAnsiChar';

type
   TCallbackNativeIRData = procedure(IRCode : TIRData); stdcall;

type
   TCallbackPAnsiChar = procedure(Protocol, Address, Command, Flags : PAnsiChar); stdcall;

type
   TIRData = packed record
      Protocol : byte;
      Address : word;
      Command : word;
      Flags : byte;
   end;
```

C++ Builder / Microsoft Visual C++:

```
#define USBIRRemoteReceiver    "USB_IR_Remote_Receiver.dll";

const char * __stdcall Version();
const char * __stdcall Copyright();
const char * __stdcall PluginName();
void __stdcall ShowSettings( HWND Handle );

BOOL __stdcall InitNative( CallbackNativeIRData * Callback );
BOOL __stdcall InitPAnsiChar( CallbackPAnsiChar * Callback );

typedef void ( __stdcall * CallbackNativeIRData) ( IRMP_DATA IRCode );
typedef void ( __stdcall * CallbackPAnsiChar) ( char * Protocol , char * Address , char * Command , char * Flags  );

typedef struct
{
   byte    Protocol;
   word    Address;
   word    Command;
   byte    Flags;
} IRMP_DATA;
```

---

function **Version**():**PAnsiChar**; stdcall external *USBIRRemoteReceiver* name 'Version';

Function will return the plugin version as PAnsiChar.

Parameters

none

Return values

Function will return USB IR Remote Receiver DLL version as PAnsiChar.

---

function **Copyright**():**PAnsiChar**; stdcall external *USBIRRemoteReceiver* name 'Copyright';

Function to get copyright of USB IR Remote Receiver DLL.

Parameters

none

Return values

Function will return copyright as PAnsiChar.

---

function **PluginName**():**PAnsiChar**; stdcall external *USBIRRemoteReceiver* name 'PluginName';

Function to get USB IR Remote Receiver DLL plugin name.

Parameters

none

Return values
Function will return USB IR Remote Receiver DLL plugin name as PAnsiChar.

---

procedure **ShowSettings**(***Handle*** : **THandle**); stdcall external *USBIRRemoteReceiver* name 'ShowSettings';

Procedure to show options/settings dialog of USB IR Remote Receiver DLL.
Parameters
Handle
[in] Window handle of the calling program. The dialog will be shown modal.

Notes:
The options/settings dialog will only be shown if the DLL got initialized first by *InitNative* / *InitPAnsiChar*.

---

function **InitNative**(***Callback*** : **TCallbackNativeIRData**):**boolean**; stdcall external *USBIRRemoteReceiver* name 'InitNative';

Function to init USB IR Remote Receiver for native callback.
Parameters
Callback
[in] Procedure pointer of a procedure of type struct TCallbackNativeIRData.

Return values
If the function succeeds, the return value is *True*.
If the function fails, the return value is *False*.

Notes:
The device get only opend if the *Callback* procedure pointer isn't *NIL / NULL*.
For closing the device, unloading/disable the plugin call this function again with *Callback = NIL / NULL*.

The defined Callback procedure will receive the IR codes like the type struct TIRData.

---

function **InitPAnsiChar**(***Callback*** : **TCallbackPAnsiChar**):**boolean**; stdcall external *USBIRRemoteReceiver* name 'InitPAnsiChar';

Function to init USB IR Remote Receiver for native callback.
Parameters
Callback
[in] Procedure pointer of a procedure of type struct TCallbackPAnsiChar.

Return values
If the function succeeds, the return value is *True*.
If the function fails, the return value is *False*.

Notes:
The device get only opend if the *Callback* procedure pointer isn't *NIL / NULL*.
For closing the device, unloading/disable the plugin call this function again with *Callback = NIL / NULL*.

The defined Callback procedure will receive the IR codes as *Protocol, Adress, Command* and *Flags* : **PAnsiChar**.

For more information see also the demo application.