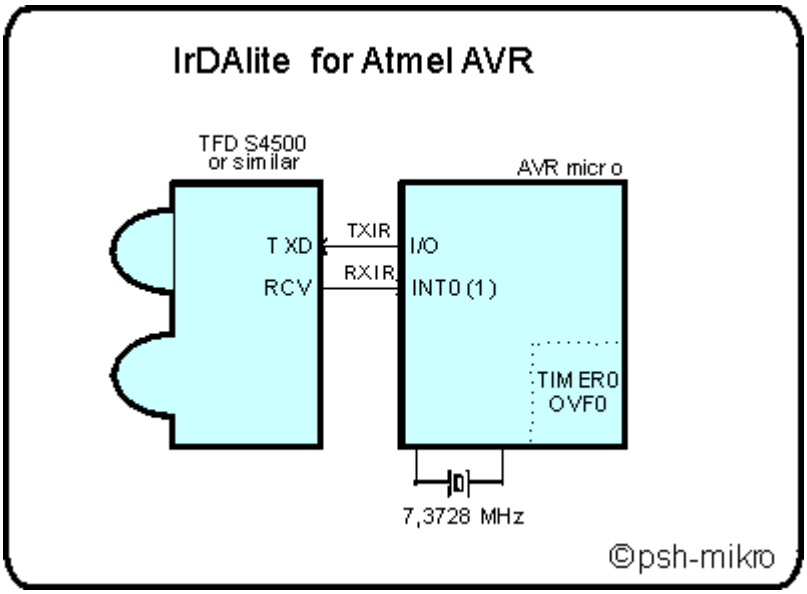
IrDA lite protocol stack for Atmel AVR microcontrollers

Features:

IrDA lite protocol fully implemented in firmware (only IR-transceiver required)

- secondary only device
- fixed speed of 9600 bps, data size =64bytes, window size =1, max. turn arround time 500 ms, disconnect/treshold time of 3 seconds, additional BOFs =0
- implemented layers: physical, frame wraper, payload and LAP , LMP , TinyTP , IrCOMM and IAS



used

- one external interrupt input, one general I/O pin as output and one 8-bit timer for timing
- ca XX Byte of SRAM
- ca 1 kByte of Program FLASH

written in AVR Studio 4.11 (assembler)

The IrDA Lite was developed for simple electronic devices with small system resources. For IrDA lite does not exist a defined specification. A Lite implementation can range from a minimum Lite up to a complete IrDA implementation. IrDA Lite is compatible with a fully implemented IrDA stack

---

Main program loop

Initialization

- clearing of SRAM and register

- set port D that on INT0 is the input

- set timer TIMER0, enable interrupt overflow

- set and enable external interrupt at the falling edge of INT0

- UART set only for the example, otherwise it is not needed

- initialization of the IrDA subsystem

The main program loop is very simple. It is only required to check following flags:

- **timeOut** might happen during IrDA communication when the connection is lost. If it appears it is necessary to go to the IrDA subsystem handling. Depending on the current stage, it terminates the connection or at least goes to the stage of a disconnected mode.

- **ENDTX**, is set if a character sent by the IR sender is sent. Based on that the next character of the sent frame can be sent or change to the receive mode if all characters of the frame are sent. This all is handled by the IrDA subsystem through calling the irtxempty procedure.

- **BYTERX** is set by an interrupt when the receiving of a character is finished. It is stored in the variable RXData a moved into the IrDA subsystem through calling the IrRxChar procedure.

- **RXCOM** is set if in the COM_buffer are data stored which are received in the valid data frame. The length of received data is stored in COMCountRxData. The user should process them now. If the user decides to send data back so the data need to put into the ir_out_buffer.
  In AppTxCount has to be set the count of bytes to be sent (max 62 in one paket). Important to store first the data into the buffer and afterwards to set the count. A count not equal zero in AppTxCount is the sign for the application layer to sent these date at the next possible occasion.

- **TXCOM** is set when all data are successfully sent and next data could be sent if needed.

Callback functions

The IrDA subsystem is calling a number of callback functions of the user program.

- **SendDataToCOMM** - is always called when data from the irCOM layer is delivered which means that a data character arrives. 62 of these character can arrive consecutively. But it is not known if these are all valid characters. Therefore the control sum is at the end of the data frame. In the example it is stored in the input buffer ir_in_buffer.

- **DataToCOMMValid** - is called in the moment when proven by the control sum all received

data are valid. The example program copies them into the COM_buffer and advise the main program to process them. It would be enough to communicate only that the data are valid a the main program would find them in the ir_in_buffer. But copying them allows to process them while already receiving new data.

- **DataToCOMMError** - indicates that data stored in the input buffer are not valid, because the control sum of the frame is incorrect. In the example nothing is copied and nothing is reported to the main program. LAP is requesting the re-send of this data and no actions necessary from the user program.

- **COMTxDataStart** is asking if data will be sent by the application. It is called if the primary device allows to sent data. In the example it is checked if in the output buffer are data. If the value in AppTxCount is unequal zero than the data transfer to the primary device is started.

- **COMTxData** - is asking for data to send. The user program is sending back a character and at the same time indicating by ZF (ZERO FLAG) if more characters are following or not. If ZF=0 (means more data) this function is repeated (max 62 times) and all the data are sent in one data frame. If ZF=1 it indicates that it is the last data and the sending of this fram is finished. In the example all data stored in the ir_out_buffer are sent consecutively.

- **COMTxValid** is calling when the primary device is confirming that the data are received. In the example the flag TXCOM is set, so that the main loop get to know that another data frame could be sent.

- **COMTxError** is following after sending the data frame if the primary device doesn't confirm that the data are successful received. In the example it is solved in a way that it shows again to the beginning and data are re-send.

Interrupts

The start bit of the serial infrared data stream triggers the external interrupt INT0. This interrupt is disabled and the timer starts measuring the single bits. When timer overflow it is looked for the pulse. Pulse means logic 0 and no pulse means logic 1. After one character is received, it is moved on and the external interrupt INT0 is enable again.

When sending the TIMER0 is used to measure the length of the single bits. External interrupt INT0 is disabled to avoid reflections. The flag TXBYTE is set what means that the sending is finished.

IrDA subsystem

It is not recommended that the IrDA subsystem is modified by the user. For inquiries and questions please contact irdavr@psh-mikro.sk

---

Implemented example

For an easy start an example is implemented. After PON Reset AVR sends the string "IrDA Demo V.2.04 11.04.2005" on the serial port (115,2k @ 7, and 3, 6864 MHz, 9600 @ 2.0000 MHz).

IrDA communication

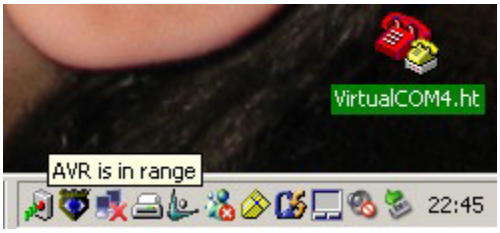
For PalmOS PDA you can use a terminal programm like "Online" from www.marktspace.com

Select communication over infrared, 9600 Baud, 8N1N and open port. Then type "r" and you should get back a memory dump of internal AVR memory space 0x200-0x210.

The AVR is confirming by sending back the received character over the serial port.

The same can be tested with a Windows based PC.

In the taskbar you should be seeing



To get a IrCOMM connection you must have running a program like hyperterm that accesses the COM emulation of IrCOMM. For virtual COMM driver for winXP see: http://www.ircomm2k.de/

Useful links:

IrDA related resources:

- IrDA Serial Infrared (SIR) Data Specifications
- IrDA Control Specification (Formerly IrBus)
- IrDA Infrared Communications Protocol (IrCOMM)1.0
- IrDA Infrared Tiny Transport Protocol (TinyTP) 1.1
- IrDA Minimal IrDA Protocol Implementation (IrDA Lite)
-

AVR related resources:

- http://www.atmel.com/products/avr/
- http://www.avrfreaks.net/
-

Palm related resources:

- PalmOS-SDK, Memory and Communications Management
- IR Ping - IrDA analyzer and demo application for IrDA programming
-

Windows related resources:

- http://www.ircomm2k.de/ - Virtual Infrared COM Port for Windows 2000/XP