WiBo - The Wireless Bootloader

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
	2010/07		DT

Contents

Abstract

The following article describes the usage of the uracoli wireless bootloader. WiBo works like a regular bootloader, except that it uses the radio transceiver instead of a UART.

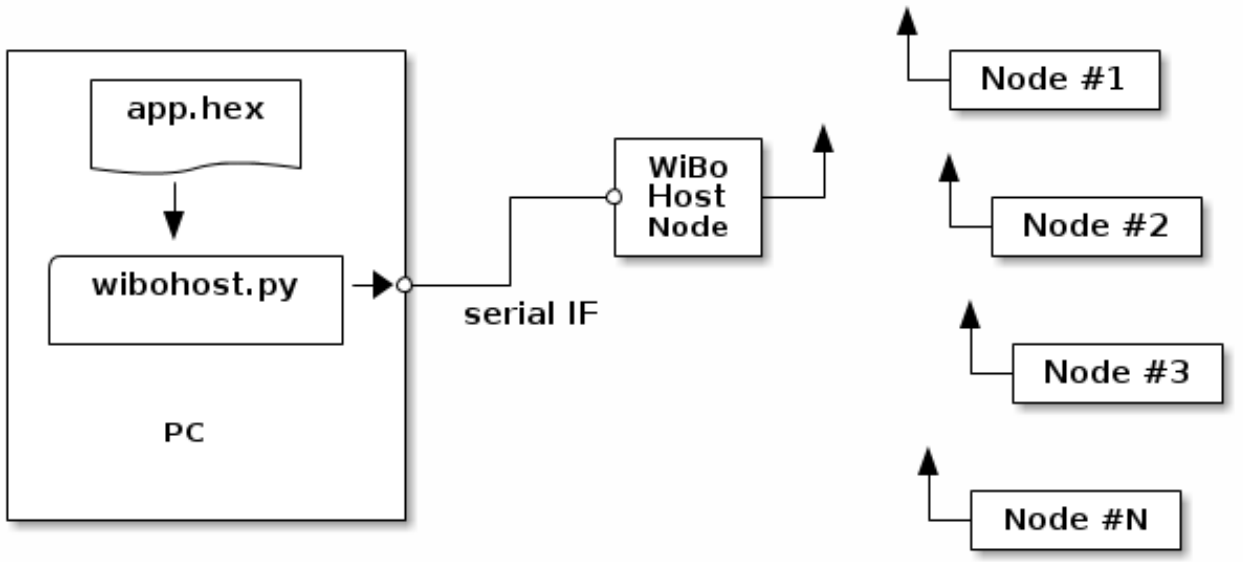
1 Concept

The WiBo framework provides a method to wirelessly flash AVR MCUs. Three components are involved,

- a PC with USB or serial interface, running the script wibohost.py,
- a host node, running with the wibohost firmware and
- the network nodes that have the bootloader installed.

The python script `wibohost.py` sends a hexfile `app.hex` in slices to the host node. The firmware of the host node transmits these data slices as unicast or as multicast frames to the network nodes. The network nodes collect the data slices and programm them with SPM (self programming mode) commands in the flash application section of the microcontroller.

In order to verfify if a node was programmed correctly or not, the host and network nodes calculate a CRC16 over all programmed data. The wibohost script can query the CRC from the nodes and verify the programming integrity.



The bootloading can be done either in **unicast** mode or in **broadcast** mode. The unicast mode can be compared to with the normal programming of MCUs, because one node is programmed with image to a time. In the broadcast mode, multiple nodes receive one image at the a time. This make sense, if the nodes consist of the hardware and each node shall run the same firmware.

The host site consists of a Python script and the host node that runs the wibohost firmware. The script `wibohost.py` uses the module pyserial for serial communication with the host node. It reads and parses the given hex file and transfers it in slices to the host node. The host node firmware sends this slices in frames to one (unicast) or all (broadcast) network nodes.

The wireless bootloader resides in bootloader section of the network nodes and occupy about 1K words of programm memory. Additionally to the bootloader code, configuration record at the end of the bootloader section. This record ensures that the node address and channel information is available, even if the EEPROM was accidentely erased.

The network nodes are passive, that means they never send anything if not queried by the host node.

2 Quick Start

1. Compile the firmware images wibo_<node>.hex, wibohost_<host>.hex, xmpl_wibo_<node>.hex

2. Flash wibo_<node>.hex together with the configuration record to the network nodes.
3. Set "BOOTRST" fuse on the network nodes to jump per default into bootloader.
4. Flash wibohost_<board>.hex on the host node
5. Connect the host node with the PC
6. Run the programm wibohost.py on the PC and e.g. flash all nodes with a app.hex
   `python wibohost.py -P <SPORT> -A 1:16 -U app.hex`

3 The Bootloader Application

Build and Flash Build the bootloader for your board with the command

```
make -C ../src <board>
make -f wibo.mk <board>
```

With the command `make -f wibo.mk list` the available <board>s are displayed.

The bootloader expects an address record at the end of the flash memory section. This record can be generated with the script nodeaddr.py. Here is an example for the rdk230 board.

```
# generate a hex file with the configuration record for node #1
python nodeaddr.py -a 1 -p 1 -c 11 \
       -f ../bin/wibo_rdk230.hex -B rdk230 -o a1.hex

# flash node #1 (SHORT_ADDR=1)
avrdude -P usb -p m1281 -c jtag2 -U a1.hex

# Fuses for initial jump to bootloader
avrdude -P usb -p m1281 -c jtag2 -U lf:w:0xe2:m \
        -U hf:w:0x98:m -U ef:w:0xff:m
```

To verify the correct AVR fuse settings refer to http://www.engbedded.com/fusecalc.

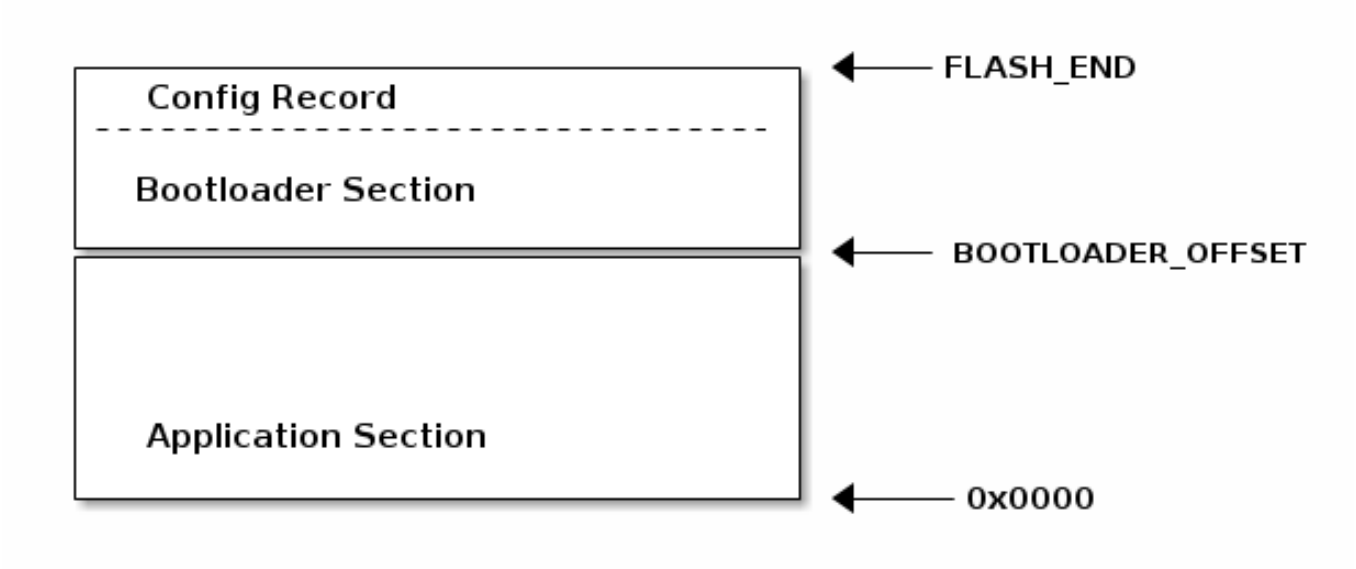
To flash multiple nodes more efficiently, the nodeaddr.py can pipe its output directly into avrdude. This slightly more complex command line can be stored in script `flashwibo.sh` (under Windows replace $1 by %1 for `flashwibo.bat`):

```
python nodeaddr.py -a $1 -p 1 -c 11 -f ../bin/wibo_rdk230.hex -B rdk230 |\
       avrdude -P usb -p m1281 -c jtag2 \
                -U fl:w:-:i -U lf:w:0xe2:m -U hf:w:0x98:m -U ef:w:0xff:m
```

Note: The option "-p 1" set the IEEE PAN_ID to "1" and must be identical for the bootloader application and the wibohost application.

With the `python nodeaddr.py -h` the help screen is displayed.

Flash memory partitioning The AVR flash memory can be divided in an application and a booloader section. The application section is located in the lower address memory. The bootloader section is located in the upper flash memory. In this section the so called self programming opcodes (SPM) can be executed by the AVR core. This SPM opcodes allows erasing and reprogramming the application flash memory.



The start address of the bootloader section is determined by the BOOTLSZ fuse bits. The BOOTRST fuse bit determines, if the AVR core jumps after reset to the application section (address 0x0000) or to the bootloader section (e.g. address 0xf000). The AVR fuse bits and the content of the bootloader section can only be changed either by ISP, JTAG or High Voltage programming.

In order to have enough memory for the application available, the booloader section is choosen to be rather small, e.g. for 8K devices a 1K bootloader section and 7K application section is a reasonable choice. The larger 128k AVR devices are partitioned usually with a 4K bootloader section, leaving 124K flash memory for the application.

The Configuration Record at FLASH_END For WiBo, the last 16 byte of the flash memory are reserved for a configuration record, that holds address and channel parameters, which are needed for operation. The structure of this record is defined in file `board.h` in the type `node_config_t`. It stores

- 2 byte SHORT_ADDRESS
- 2 byte PAN ID
- 8 byte IEEE_ADDRESS
- 1 byte channel hint
- 2 reserved bytes
- 2 byte CRC16

```
The configuration record is accessible from the applicatation and from the bootloader sec
```

4 The Host Application

Build and Flash Here is an example for the Raven USB Stick. Applying the configuration record to the hex-file is done in the same manner as for the bootloader application.

```
cd wibo
make -C ../src rzusb
make -f wibohost.mk rzusb
python nodeaddr.py -a 0 -p 1 -f ../bin/wibohost_rzusb.hex -B rzusb -o h.hex
avrdude -P usb -c jtag2 -p at90usb1287 -U h.hex
```

Note: The option "-p 1" set the IEEE PAN_ID to "1" and must be identical for the bootloader application and the wibohost application.

5 Python Host Application

Using wibohost.py To test the wireless bootloader environment, the xmpl_wibo application will be used. It blinks a LED with a certain frequency and is able to jump in the bootloader when the special "jump_to_bootloader" frame is received.

At first create some firmware versions, e.g. one slow and one fast blinking. The network nodes shall be rdk230 nodes.

```
make -f xmpl_wibo.mk BLINK=0x7fffUL TARGET=slow.hex rdk230
make -f xmpl_wibo.mk BLINK=0xffffUL TARGET=fast.hex rdk230
```

Next assume that you have 4 network nodes with addresses [1,2,3,4]. In order to check the presence of the nodes, run the scan command.

```
python wibohost.py -P COM1 -S
```

Note that the default address range of wibohost.py is 1 ... 8. This can be modified with the -A option. In the example above, only the nodes 1 to 4 are present, therefore no response from the nodes 5 ... 8 is received.

At first we update all nodes with the slow blinking firmware. Therefore we use the broadcast mode (-U), that means that the image is transfered only once over the air. The address range (-A) is needed to ping the nodes before programming and afterwards to verify their CRC.

```
python wibhost.py -P COM1 -A1:4 -U slow.hex
```

In the next step we selectively flash node 1 and node 3 with the file fast.hex. Since we use unicast programming (-u), the image is transfered for each node over the air seperately.

```
python wibhost.py -P COM1 -A1,3 -u fast.hex
```

The WiBoHost API The file wibohost.py can also be used as a python module. The following script shows how a broadcast and a unicast flash can be programmed.

```
from wibohost import WIBOHost

wh = WIBOHost()
# open serial connection to wibohost node
wh.close()
wh.setPort("COM19")
wh.setBaudrate(38400)
wh.setTimeout(1)
wh.open()
# check if local echo works.
print WHOST.echo("The quick brown fox jumps")

# scan addresses 1 to 4
addr_lst = wh.scan(range(1,4+1))

# broadcast mode, flash all nodes
for n in addr_lst:
    wh.xmpljbootl(n)
    print "PING  :", n, wh.ping(n)

print "FLASH :", wh.flashhex(0xffff, "foo.hex")
for n in addr_lst:
    print "CRC  :", n, wh.checkcrc(n)
    print "EXIT :", n, wh.exit(n)

# unicast mode, flash node 1
wh.xmpljbootl(1)
print "PING  :#1", wh.ping(1)
print "FLASH :#1", wh.flashhex(1, "bar.hex")
print "CRC   :#1", wh.checkcrc(1)
print "EXIT  :#1", wh.exit(1)
```