

Contiki OS port for ez430-RF2500

Entry to the MSP430 Design Contest

Wincent Balin

Abstract: ez430-RF2500 is an affordable development kit with a MSP430 controller and a radio chip, which allows to connect between two boards supplied in the kit. The aim of this project is to port Contiki OS to the aforementioned kit to employ it as a smallest scale network, which, using one kit only, consists from just an access point and one node. Contiki is an operating system for memory-constrained embedded system with built-in IP networking. It is possible to utilize the ez430-RF2500 development kit as a remotely controlled multi-purpose embedded system, as both the access point and the node have free and accessible digital in- and outputs. The system is quite low-power as it spends most of the time in low-power mode LPM4.

Table of contents

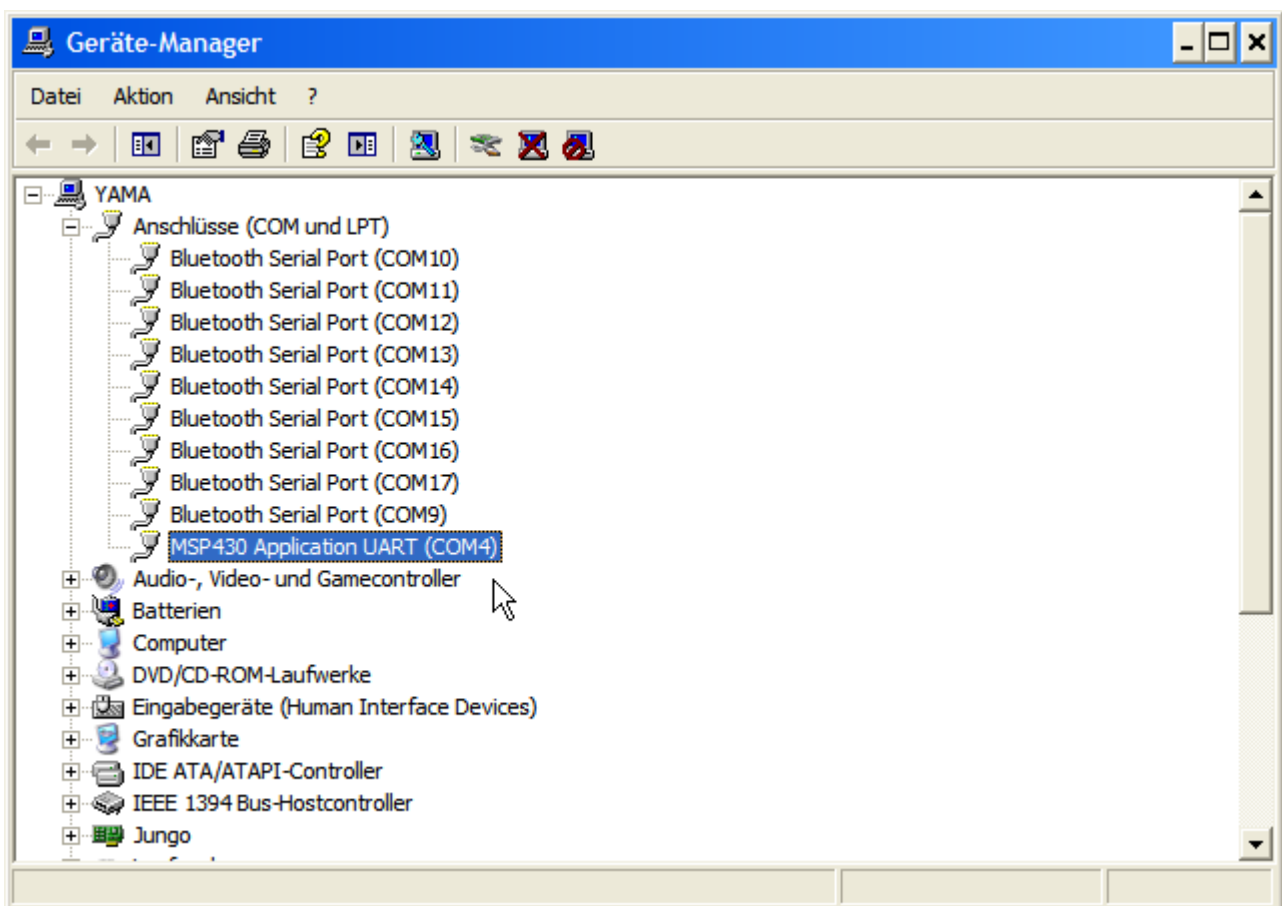
Development tools.....	4
Installation of the serial port driver.....	4
Port of the Contiki OS.....	4
Testing application.....	6
Known peculiarities.....	7

Development tools

The development was done in Windows XP using Cygwin environment. All files were edited with Crimson Editor. The Contiki OS was compiled using MSPGCC 3.2.3 20081230. The debugging was done utilizing GDB supplied with MSPGCC package, which in turn used the msp430-gdbproxy tool for communication with the development kit. When needed, Windows-supplied terminal program was used for serial communication with boards.

Installation of the serial port driver

If you have not install the drivers for ez430-RF2500 kit, they can be found in the directory `contiki-2.3-rf2500/tools/rf2500`. You will be asked about the driver when you connect the USB emulator board to your computer for the first time. Pay attention about the name of the serial port the driver creates. Look into the Device Manager to find it. In the figure below it is COM4.



Port of the Contiki OS¹

The About Contiki page² states that "Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks". Because the boards supplied with ez430-RF2500 development kit have both a MSP430 microcontroller, to which Contiki has been ported already, and the radio chip of the family that is in

1 <http://www.sics.se/contiki/>

2 <http://www.sics.se/contiki/about-contiki.html>

use on most of the wireless node platforms Contiki has been already ported to, the idea arised to port Contiki to ez430-RF2500 (further in the text RF2500) nodes.

The difficulties met had the cause of previous porting of Contiki to MSP430F1XX platforms with at least 2 kb RAM. The controllers used on RF2500 nodes have 1 kb only. Because of that another kind of a network than TCP/IP had to be implemented. Another difficulty was to get button sensor³ working. The cause here is the usage of API which is working with MSP430F1XX controllers, but apparently not with MSP430F2XX.

The CC2500 radio chip has a driver now, which uses driver implementation other that Contiki canon one. There is a good possibility either to make a CC2400 compatible driver (CC2500 has a CC2400 compatibility bit) or to re-implement a driver after the Contiki convention (look at the existing CC2420 driver, for example). The current driver is implemented in the testing application (see below).

It was very useful for the whole development process to have a previously written application which uses MSPGCC in stead of IAR compiler. This application, as well as a MSPGCC port of example Simpliciti application supplied with ez430-RF2500, was created by Jean-Marc Koller⁴.

Currently it is possible to measure battery voltage, temperature, wireless link quality and signal strength. It is possible to transfer the measurements to another node. The access point node runs Contiki shell over serial line. The shell has command `lsensors` for local sensor measurements and `rsensors` for remote sensor measurements.

If you will look into the archive you downloaded, which also contains this document, you will find the platform part of the port in the directory `contiki-2.3-rf2500/platform/rf2500`. Below is the explanation of the files you will find therein.

`clock.c` is the Contiki clock driver. It is a rather straight port from MSP430F1XX kind.

`contiki-conf.h` is the hardware configuration file. You will find there most of the hardware settings that matter.

`contiki-rf2500-main.c` is the system starter with `main()` function. It also starts most of the Contiki services.

`leds-arch.c` is the architecture-dependent implementation of LED API of Contiki.

`Makefile` is a make file which starts compilation of the platform files.

`Makefile.rf2500` has the settings needed to compile the rf2500 platform.

`rtimer-arch.c` is the architecture-dependent implementation of the rtimer API of Contiki.

`rtimer-arch.h` is the architecture-dependent specification par of the rtimer API of Contiki.

`spi-arch.c` is the architecture-dependent part of SPI API of Contiki. Actually, in this port only SPI initialization is used.

The `dev` directory contains driver files, which are listed below with explanations.

`adc-sensors.c` is the driver for both battery voltage sensor and the temperature sensor. Both utilize ADC core in the microcontroller.

³ Contiki API for digital inputs with manual switch

⁴ <http://jmkikori.homepage.bluewin.ch/devel/ez430-rf2500.html>

Look at the `raw_tranceiver` application at the page.

`adc-sensors.h` is the include file for the `adc-sensors` driver.

`button-sensor.c` is the driver for button sensor.

`cc2500_regs.h` was taken from the aforementioned application by Jean-Marc Koller. Same goes for `smartrf_CC2500.c`. Both files were created by Texas Instruments.

`irq.c` is the MSP430F2XX reimplementation of `contiki-2.3-rf2500/cpu/msp430/dev/irq.c`.

`msp430.c` is the MSP430F2XX reimplementation of `contiki-2.3-rf2500/cpu/msp430/msp430.c`

`radio-sensor.c` is the driver for radio sensor (signal strength etc.)

`serial-line.c` is the minimally changed driver for serial line. It solves the issue with `process_poll()` function (see in the **Known peculiarities** chapter).

`uart1.c` is the MSP430F2XX reimplementation of `contiki-2.3-rf2500/cpu/msp430/dev/uart1.c`

Testing application

You will find the testing application in the directory `contiki-2.3-rf2500/examples/rf2500jmk`. The files belonging to the application are listed below and explained.

`gdb-commands` is an auxiliary file, so that the developer might upload application and start GDB using command `msp430-gdb -x gdb-commands rf2500jmk.rf2500`.

`Makefile` is the application specific Makefile.

`rf2500jmk.c` is the application source file.

`rf2500jmk.rf2500` is the ready-to-install binary for both RF2500 nodes.

`serial-shell.c` is a correction for Contiki-supplied serial shell. Corrected shell does not read address values from the (also Contiki-supplied) RIME stack, which could be used because of memory constraints.

`shell-lsensors.c` is the implementation of the `lsensors` shell command.

`shell-lsensors.h` is the header file of the `lsensors` shell command.

`shell-rsensors.c` is the implementation of the `rsensors` shell command.

`shell-rsensors.h` is the header file of the `rsensors` shell command.

The application resides, as mentioned above, in the file `rf2500jmk.c`. After the system processes are set up, both main and beacon processes start through Contiki Autostart API. Beacon process sends a beacon string every 2 seconds. Main process waits for a frame received. After a correct (verified by CC2500 built-in CRC) frame was received, the process checks it's contents. If a "rsensors" string (`rsensors` command request) was received, the process encodes the information from it's sensors and sends it back. If a string beginning with "RS" (`rsensors` command reply) was received, the sensor information in it is decoded and printed on the serial terminal. Else, a beacon string is sent back after a 0.25 seconds. The latter behaviour results in a constant ping-pong between two nodes (and therefore a constant LED blinking) and works as a radio connection indicator.

On the side of the access point you may connect with a serial terminal program to it. Connect to the serial interface you looked up above in the chapter **Installation of the serial port driver**. Use following settings: 9600 baud, 8 data bits, no parity, 1 stop bit, no handshake; local echo. Type "?" or "help" to list possible commands. Type "lsensors" to get data from local sensors. Type "rsensors" to get data from remote sensors. An example output can be found in the figure below.

```
Contiki-RF2500 - HyperTerminal
Datei Bearbeiten Ansicht Anrufen Übertragung ?
Contiki>
lsensors
Values of local sensors:
Button sensor: off
Battery sensor: 3.5 V
Temperature sensor: 32 Celsius
Radio sensor: LQI=18 RSSI=39
Contiki>
rsensors
Send command...
Contiki>
Values of remote sensors:
Button sensor: off
Battery sensor: 2.8 V
Temperature sensor: 17 Celsius
Radio sensor: LQI=16 RSSI=38
Contiki>
-
Verbunden 00:01:04 VT100 9600 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho
```

Known peculiarities

When you use `PROCESS_YIELD()` in the RF2500 port, the polling of the process with the `process_poll(&process_name)` works only when the function with `process_poll()` is implemented before process `process_name`. Otherwise use `process_post_event(&process_name, PROCESS_EVENT_POLL, 0)`.

Button sensor is not really working. This can be corrected if one implements button sensor driver without using `HWCONF` API of Contiki. The mentioned above `raw_tranceiver` application shows how to do it. If done so, one can get rid of the `contiki-2.3-rf2500/platform/rf2500/dev/irq.c` file too.

The `AUTOSTART_PROCESSES` macro seems to work only if all autostart processes are in the same file.