VirtualSense User Guide 1

Environment Installation

University of Urbino & NeuNet
www.virtualsense.it
Revision 1.0

May 16, 2013

This guide applies to VirtualSense hardware platform 1.1.0.

Contents

List of Figures

1 Introduction

The availability of off-the-shelf micro controller units based on energy efficient 16-bit RISC processors which provide a wide range of low-power inactive modes with average current in the range of micro Watts and wake-up times in the range of micro seconds makes it possible to develop ultra-low-power sensor nodes able to run a virtual machine to speedup the development and the deployment of sensing/monitoring applications.
VirtualSense is an open-hardware/open-source project which aims at the development of IEEE 802.15.4-compliant low-cost ultra-low-power wireless sensor nodes providing a Java-compatible runtime environment which grants to the programmer full control of the low-power states of the hardware.

2 General description and Key Features

VirtualSense is an ultra-low power wireless node for use in wireless sensor networks (WSNs) subject to tight power constraints. Thanks to the on board Java compatible virtual machine (VM) it allows programmers to rapidly develop monitoring applications and communication protocols. VirtualSense makes use of IEEE 802.15.4 wireless transceivers in order to standardize communication and to inter-operate with other existing devices. The set of on board sensors (including pressure, temperature, and light), together with the possibility to easily connect any external sensor/actuator, allows VirtualSense to be used in a wide range of application fields.
In order to promote research and development VirtualSense adopts an open-hardware/open-source model. In particular, it mounts widely available off-the-shelf components and it makes publicly available all PCB schematics. The open-source software stack is based on a modified version of Darjeeling java-compatible VM running on top of Contiki operating system.

3 Key components and features

The key components of VirtualSense 1.0 are listed below:

- 250kbps 2.4GHz IEEE 802.15.4 Texas Instruments cc2520 Wireless Transceiver
- 25MHz Texas Instruments MSP430f54xxa microcontroller unit (MCU) with 16k RAM and 128k Flash
- Integrated Humidity, Temperature, and Light sensors
- 512K I2C™ Serial EEPROM
- On-board 48-bit I2C Extended Unique Identifier (EUI-48™)
- On-board programmable ultra-low-power RTC

The distinguishing features include:

- Ultra low power consumption ($\approx$10W in hibernation, $\approx$100W in sleep mode, 50/60mW in send/receive modes, respectively)
- with state-of-the-art energy harvesting modules
- Fast wakeup from sleep mode ($<$5s)
- Programmable timed wake-up from any low-power mode
- Sensitivity to asynchronous external events
- Integrated 12-bit ADC/DAC
- Integrated Supply Voltage Supervisor (SVS)
- Integrated DMA Controller
- USB 2.0 RS232/UART communication with a PC
- Interoperability with other IEEE 802.15.4 devices
- Open-source software stack
- Contiki MAC-layer compatibility
- Java-compatible run-time environment
- Easy Over the Air (OTA) programming

4 Architecture

VirtualSense is made of ultra-low-power components in order to keep the average consumption compatible with state-of-the-art energy harvesters. Figure 2 shows the functional block diagram representing the node architecture. The core is a MCU belonging to the Texas Instrument MSP430F54xxa family. It communicates through I2C™ bus with a Microchip 24AA025E48 Extended Unique Identifier and with a Microchip 24AA512 serial 512K EEPROM. Using the SPI bus, the MCU manages the Texas Instruments CC2520 2.4GHz IEEE 802.15.4 RF transceiver and communicates with the NXP PCF2123 ultra low-power real time clock/calendar.

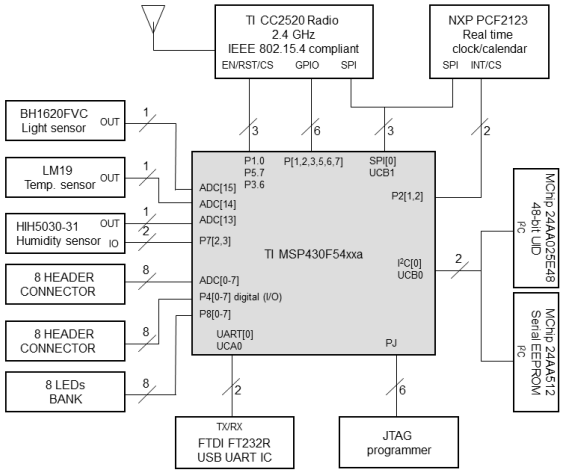

Figure 1: The functional block diagram

5 Hardware and software stacks

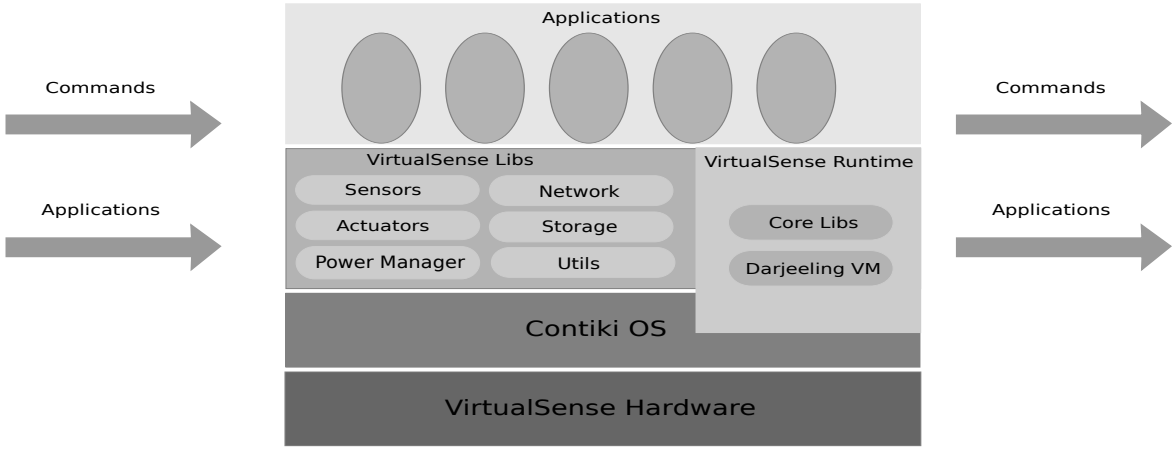

Figure 2: Hardware and software staks

At the lowest stack level there is VirtualSesnse hardware that provides to higher levels: an network interface, an ultra-low-power microcontroller and a set of sensors. In addition to ensuring low power consumption, the 20-bit processor on the VirtualSense, allows hight performance so that at the base of software stack of VirtualSense can be an operative system like CONTIKI. The great versatility of CONTIKI permit to run on a processor with 16k RAM and 128k Flash, a real virtual machines like the JVM that allow usage an hight-level programming language like Java™. The VirtualSenese Runtime represents the most important stack level, because provide to powered application leaving the approach near-hardware to using a simple set of API (described in VirtualSense Libs) that give access to all hardware functionality. The high abstraction of VirtualSense runtime architecture provide to create multi-threading applications that can run concurrently on a single node increasing the use cases of VirtualSense. Because the apps running on Virtualsense are not on the firmware but in a higher level, them can be removed, updated or added by remote only sending some special package and commands on the network. The hight performance and great versatility of VirtualSense not affect the power consumption, thanks to hight optimization of VirtualSense runtime architecture, that takes full advantage of the low-power state of MSP430 microcontrollers.

6 Installation environment

For getting started to work on VirtualSense you must install the environment. VirtualSense has been developed on a Linux like OS, in this guide will be explained how to install the environment on a Linux distribution (downloaded from http://www.ubuntu-it.org/download) more precisely has been used a distribution of Linux **Lubuntu 12.10**. The first step is install the mspgcc tools (binutils, gcc, gdb and a lot of other tools for the MSP430 processor).

6.1 Install MSPGCC tools

Open the **Terminal** (Alt + Ctrl + t) and insert the following commands:

Create a temporary **build directory** on your home.

```
mkdir msp430-build
cd msp430-build
```

Download current development version of **msp430gcc** (in this case is 20 bit mspgcc-20120911) from http://sourceforge.net/projects/mspgcc/files/mspgcc/DEVEL-4.7.x/ and unpack the package in the build directory.

```
wget http://sourceforge.net/projects/mspgcc/files/mspgcc/DEVEL-4.7.x/mspgcc
   -20120911.tar.bz2
tar xvfj mspgcc-20120911.tar.bz2
```

This **package** contains patches necessary to update the mainstream GNU files version and some other file which allow to select the correct version of mspgcc tool about the downloaded version.

```
~/msp430-build$ cd mspgcc-20120911
~/msp430-build/mspgcc-20120911$ ls

msp430-binutils-2.22-20120911.patch      msp430mcu.version      htdocs
msp430-gcc-4.7.0-20120911.patch          README                 docs
msp430-gdb-7.2a-20111205.patch           README.rst
msp430-libc.version                      RELEASES.TXT
```

The packages to download are:

- **msp430mcu** - http://sourceforge.net/projects/mspgcc/files/msp430mcu/
- **minimal libc** - http://sourceforge.net/projects/mspgcc/files/msp430-libc/
- **binutils** - http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/binutils/
- **GCC** - http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/gcc/gcc-4.7.0/
- **GDB** - http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/gdb/
- **mspdebug** - http://sourceforge.net/projects/mspdebug/files/

Check the **correct version** of all package and download them with command **wget**, in this case.

```
wget http://sourceforge.net/projects/mspgcc/files/msp430mcu/msp430mcu
    -20120716.tar.bz2
wget http://sourceforge.net/projects/mspgcc/files/msp430-libc/msp430-libc
    -20120716.tar.bz2
wget http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/binutils/binutils
    -2.22.tar.bz2
wget http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/gcc/gcc-4.7.0/gcc
    -4.7.0.tar.bz2
wget http://mirror2.mirror.garr.it/mirrors/gnuftp/gnu/gdb/gdb-7.2a.tar.bz2
wget http://sourceforge.net/projects/mspdebug/files/mspdebug-0.20.tar.gz
```

Extract all **downloaded packages** into the build directory.

```
tar xvfj binutils-2.22.tar.bz2
tar xvfj gcc-4.7.0.tar.bz2
tar xvfj gdb-7.2a.tar.bz2
tar xvfj msp430mcu-20120716.tar.bz2
tar xvfj msp430-libc-20120716.tar.bz2
tar xvfz mspdebug-0.20.tar.gz
```

Install the **prerequisites**.

```
cd gcc-4.7.0
./contrib/download_prerequisites
```

Install the **library necessary** and not found in lubuntu 12.10.

```
sudo apt-get install build-essential texinfo ghc gcc g++ libghc-zlib-dev
    libmpc-dev libreadline-dev libusb-dev libboost-dev libboost-all-dev
    srecord srec_cat
```

Apply the **patch** on **binutils**, **GCC** and **GDB**, using the files provided in the Release, to bring them to current release.

```
cd binutils-2.22
patch -p1<../mspgcc-20120911/msp430-binutils-2.22-20120911.patch
cd ..

cd gcc-4.7.0
patch -p1<../mspgcc-20120911/msp430-gcc-4.7.0-20120911.patch
cd ..

cd gdb-7.2
patch -p1<../mspgcc-20120911/msp430-gdb-7.2a-20111205.patch
cd ..
```

Create a **sub-set of directories** into the Build Directory for compile and install all package.

```
mkdir binutils-2.22-msp430
mkdir gcc-4.7.0-msp430
mkdir gdb-7.2-msp430
```

Configure **Binutils**, **GCC** and **GDB**.

```
cd binutils-2.22-msp430
../binutils-2.22/configure --target=msp430 --program-prefix="msp430-"

make
sudo make install

cd ..
```

If you have build issues from detect **msp430-ranlib** insert the follow commands.

```
cd /usr/bin
sudo ln -s /usr/local/bin/msp430-ranlib
```

```
cd gcc-4.7.0-msp430
../gcc-4.7.0/configure --target=msp430 --enable-languages=c --program-
   prefix="msp430-"

make
sudo make install
cd ..
```

```
cd gdb-7.2-msp430
../gdb-7.2/configure --target=msp430 --program-prefix="msp430-"
make
sudo make install
cd ..
```

Install the **mspgcc-mcu files**.

```
cd msp430mcu-20120716
sudo MSP430MCU_ROOT=`pwd` ./scripts/install.sh /usr/local/
cd ..
```

Install the **mspgcc-libc**.

```
cd msp430-libc-20120716

#  If you need to disable features, run configure here with any of the
   following flags to enable/disable features.
#  --disable-printf-int64 : Remove 64-bit integer support to printf formats
#  --disable-printf-int32 : Remove 32-bit integer support from printf
   formats
#  --enable-ieee754-errors : Use IEEE 754 error checking in libfp functions

cd src
make
sudo PATH=$PATH make PREFIX=/usr/local install
cd ..
```

Install **msp430 debugger**.

```
cd /mspdebug-0.20
make
sudo make install
```

Install **MSP debug stack** (MSPDS) that allow communication to all MSP430 microcontroller. Download MSPDS package from http://processors.wiki.ti.com/index.php/MSP_Debug_Stack, unpack the package and into the folder insert the follow commands.

```
~/Download/MSP430.DLLv3_OS_Package$

make
```

```
sudo cp libmsp430.so /usr/lib/
```

6.2 Install programming environment and utilities

Install **Eclipse** environment, **ant** compiler and a set of programming utilities useful to working on VirtualSense (git, doxygen, grapviz, cutecome).

```
sudo apt-get install eclipse
sudo apt-get install ant
sudo apt-get install git
sudo apt-get install doxygen
sudo apt-get install graphviz
sudo apt-get install cutecom
```

Download VirtualSense source code from google code repository http://virtual-sense.googlecode.com/files/VirtualSense_Platform_1.1.0.zip.

```
wget http://virtual-sense.googlecode.com/files/VirtualSense_Platform_1.1.0.
    zip
```

Unpack downloaded package.

```
unzip VirtualSense_Platform_1.1.0.zip
```

Set a **System Variable** named CONTIKI that refer at the root folder of VirtualSense platform, prefer adding it to all configuration file (bash.bashrc, profile) that will be used by Eclipse.

```
~/home/virtualsense$ sudo gvim .bashrc
```

File **.bashrc**

```
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
```

```
    fi
  fi

  export CONTIKI="/home/virtualsense/virtual-sense/VirtualSense/"  #<---
  ~
  ~
  ~
  ~
```

For safety append the row also to the file **/etc/profile** and **/etc/bash.bashrc** and update the state with the command **bash**.

```
  sudo gvim /etc/bash.bashrc
  sudo gvim /etc/profile

  bash
```

Extend the used right of serial port for the used user, adding the user at group **dialout** editing the system **group** file.

```
  sudo gvim /etc/group
```

File **group**

```
  mail:x:8:
  news:x:9:
  uucp:x:10:
  man:x:12:
  proxy:x:13:
  kmem:x:15:
  dialout:x:20:virtualsense   # <-- Insert user name here
  voice:x:22:
  cdrom:x:24:virtualsense
  floppy:x:25:
  tape:x:26:
  sudo:x:27:virtualsense
  audio:x:29:
  dip:x:30:virtualsense
```

6.3 Import VirtualSense project on Eclipse

Launch Eclipse IDE for **import VirtualSense project** previously downloaded with git command.

On Eclipse go on **File** > **Import**, select **Existing Projects into Workspace** and click **Next**.

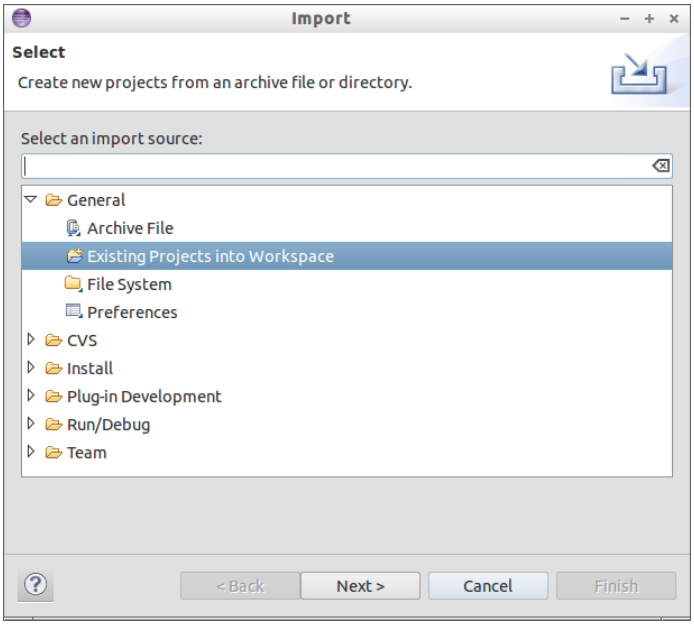

Figure 3: Eclipse project import step1

Browse the **project root directory** into directory three of VirtualSense downloaded with git and click **Finish**.

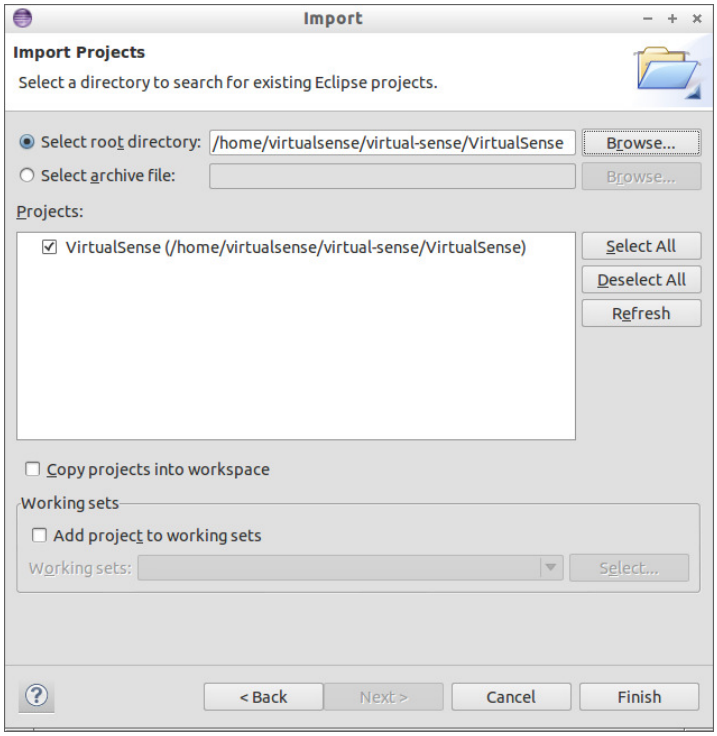

Figure 4: Eclipse project import step2

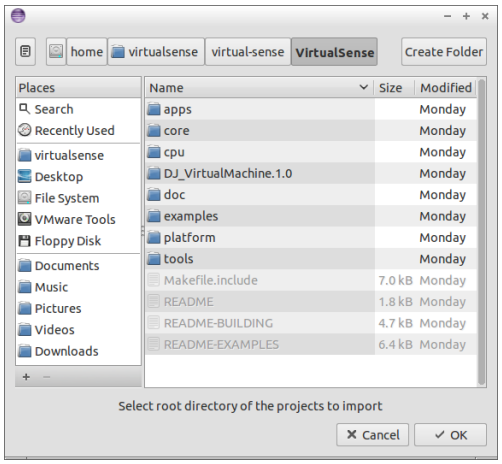

Figure 5: Eclipse browsing project folder

Now full VirtualSense project is imported on Eclipse IDE and ready to use.

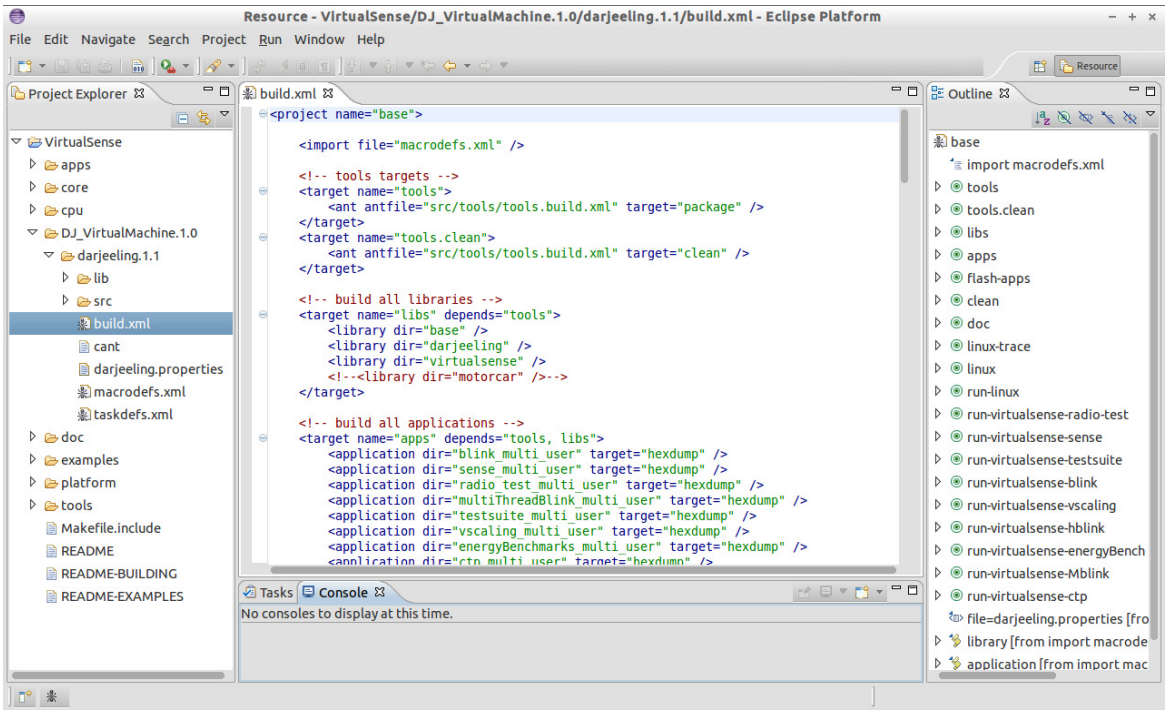

Figure 6: Eclipse project imported start view

All done, you are ready to work on VirtualSense Environment!