The Scatterweb Embedded Sensor Board

The Scatterweb ESB has been developed by the CST (Computer Systems & Telematics) group at Freie Universität Berlin (FU Berlin). It consists of a small circuit board with a microcontroller unit and sensors (sound, vibration, IR, temperature, among others), an RS232 interface and a radio transciever. The circuit board is mounted on a battery pack for three R6 batteries.

Contents

Sampling the ESB sensors

We wanted to look in detail at the data returned by the sensors, and to have detailed control over the way they were sampled, to better determine the different sensors' characteristics. To this end firmware was developed that allow sampling at 8 KiHz of any sensor (except temperature, which is sampled at 8 Hz).

esb_samplesens_fw.tgz - the sources and a pre-compiled binary (.hex file) of the firmware.
esb_samplesens.tgz - host computer program to initiate sampling of selected sensor and write the data to a file.
esb_samplesens_matlab.tgz - matlab scripts to initiate sampling of selected sensor and display sampled data in realtime (well, almost :).

Sensors supported by the firmware and host/matlab programs are:

- 8 KiHz sample rate, 8 bit samples:
  - `mic` - Microphone (analog 8-bit samples)
  - `pia` - PIR analog (analog 8-bit samples)
  - `pid` - PIR binary (data is either 0 or 255)
  - `vib` - Vibration (data is either 0 or 255)
  - `lig` - IR intensity (data is either 0 or 255)
- Other:
  - `tem` - Temperature. 16 bits per sample, where each sample is a 8.8 fixed point signed number (precision 8.1 bits). Sample rate 8 Hz (the DS1629 temp sensor requires 400 ms (typ) for each conversion, so there's no reason to sample faster).

Programming the MSP430 family on a Nada Red Hat computer

The ESB is built around the Texas Instruments MSP430F149 microcontroller.

MSP430x1xx Family User's Guide - Detailed description of the controller family and its peripherals, registers, memory configuration, Low Power Modes, clocks etc.
MSP430x13x/MSP430x14x Data Sheet - Pinouts, electrical characteristics and other device specific information.

I followed these steps to install the MSP430 develoment tools on my KTH/Nada Red Hat computer:

- Contacted the system group to get access to the local hard drive (at /NOBACKUP/my-user-name-nobackup/) and to become owner of /dev/ttyS0 and /dev/parport0.
- Followed the instructions at http://www.mikrocontroller.net/en/mspgcc, but configured everything with --prefix=/NOBACKUP/my-user-name-nobackup/msp430, and didn't move libHIL.so or run ldconfig (requires root access). Also, I did not install pyJTAG.
- Added /NOBACKUP/my-user-name-nobackup/jtag/hardware_access and /NOBACKUP/my-user-name-nobackup/msp430/lib to the LD_LIBRARY_PATH environment variable.
- Added /NOBACKUP/my-user-name-nobackup/msp430/bin to the PATH environment variable.

Programming the microcontroller

The C source files should include <io.h>. This header file defines all of the registers and flags you'll need. To transfer the binary to the mcu you need a JTAG interface - or here: JTAG interface.

- To compile C source code for the MSP430x149, use
  msp430-gcc -Os -mmcu=msp430x149 -o firmware.elf *.c
- Generating assembler listing (optional):
  msp430-objdump -DS firmware.elf > firmware.lst
- Generating hex file for flashing:
  msp430-objcopy -O ihex firmware.elf firmware.hex

I used gdb to flash the hex file to the controller. To enable gdb to talk to the JTAG programmer, run gdbproxy:

- msp430-gdbproxy --port=2000 msp430

gdbproxy connects to the microcontroller through the JTAG parallel port interface, and opens a socket at localhost:2000 for gdb to talk to.

- Add this to ~/.gdbinit (or you'll have to type it at the gdb prompt every time you flash the mcu):

```
set remoteaddresssize 64
set remotetimeout 999999
set remote memory-write-packet-size fixed
target remote localhost:2000
```

And finally, run GDB:

- msp430-gdb firmware.hex

  which makes gdb display a prompt. To *flash* your program (i.e. load you program into the nonvolatile flash memory of the mcu), type:

`mon erase`	- erase flash memory (needs to be done before you can write new data)
`load`	- store firmware in flash
`mon reset`	- reset mcu (loading f*cks with the clock, this fixes it)
`c`	- 'continue', start code execution

Running the ESB Monitor program

The ESB Monitor program supplied by the CST group display the status of the sensors and allows time etc. to be set, when the ESB is programmed with any firmware based on version 1.0 of the ESB "System" code (can be found in the archive section of the Scatterweb homepage). The program is written in Java, and to enable the JVM to talk to the serial port the RXTX package needs to be installed:

- Download the "Self-extracting binary" (not the RPM) version of J2SE 1.4.2 SDK from http://java.sun.com/ and install it by executing it (for example in the `/NOBACKUP/your-user-name-nobackup/` directory).
- Set JAVA_HOME, JAVAHOME, and JDK_HOME to point to the java directory (i.e. `/NOBACKUP/your-user-name-nobackup/j2sdk1.4.2_8`). Replace the path to the system version of java with `${JAVA_HOME}/bin` (in your PATH variable). Make sure these variables are set correctly before continuing with the RXTX installation!
- Download `rxtx-2.0-5.tar.gz` (not any newer version) from http://www.rxtx.org -> Download. Unpack and follow the INSTALL file instructions. Important: configure with --disable-lockfiles!
- Unpack the ESB Monitor source files to a subdir named `sweb`; compile from the parent dir with `javac sweb.SWeb`, run from the same location with `java sweb.SWeb`.
- Before compiling, add `"/dev/ttyS0"` (or equivalent) to the `port_Strings` array in SWebInitDialog.java.

Since a whole bunch of environment variables needs to be set or changed, I suggest that the necessary commands are written to a text file which can then be sourced, e.g.:

```
setenv JAVA_HOME /NOBACKUP/your-user-name-nobackup/j2sdk1.4.2_08
setenv JAVAHOME $JAVA_HOME
setenv JDK_HOME $JAVA_HOME
set msp430_home=/NOBACKUP/your-user-name-nobackup/msp430
setenv
PATH /usr/kerberos/bin:/pkg/netscape/7/os:/usr/local/bin:/usr/local/bin/X11:/usr/

setenv LD_LIBRARY_PATH ${msp430_home}/lib:/NOBACKUP/your-user-name-
nobackup/jtag/hardware_access
unset msp430_home
```

... and then `source <textfile>` to setup everything. (This code also sets the variables for the MSP430 development environment.) Your PATH may be different than mine, so the setenv PATH line may need to be modified accordingly.

Compiling firmware based on the OS-like "System" code

The CST Group of FU Berlin provides a "System" firmware to be used with the ESB. It initializes the ESB and provides a framework for user applications. It also handles some serial commands (e.g. `swr`: Toggle red LED). User applications can be added to the System by means of a callback function `Process_init()` that is called by the System initialization routine. The application can add timers and register callbacks for sensor events, among other things.

As of this writing, the latest version of the ESB "System" code is 2.1. It can be downloaded from the Scatterweb homepage -> ESB -> Source. When the file is unpacked two directories are created: `Applications` and `System`.

To compile the System code without any additional Application code, `cd` to `Applications/`

`[EMPTY]/ESB`, rename the `makefile Makefile`, and type `make`. The compiled .hex file can then be found in the `out` subdirectory.

A very helpful Doxygen documentation of the ESB System 2.1 source code can be found at http://www.ti5.tu-harburg.de/Staff/Witt/ESB-Firmware/main.html.

Some helpful links

- The Scatterweb homepage.
- Texas Instruments MSP430 family homepage.
- GCC toolchain for the Texas Instruments MSP430 MCUs homepage.
- Ronnie Johansson's page on the ESB, with more information about the JTAG interface, installing the development environment on other platforms, and more.
- Scatterweb.System Doxygen documentation.
- List of mcu connections on the ESB.

Created 2005-08-11, Arvid Brodin, Nada/CAS, KTH.