WSim Hardware Platform Simulator

Tutorial

COLLABORATORS

	TITLE : WSim Hardware Platform Simulator	*REFERENCE* :

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Antoine Fraboulet and Andreea Picu	February 26, 2009	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0	2007-11-07	First Draft	

Contents

List of Figures

Abstract

First try of a tutorial for WSim that covers the basics through some simple examples.

Chapter 1

Introduction

This tutorial is intended to get you started with WSim. It shows you the basics of compiling, executing, debugging and interpreting results of applications written for supported target microcontrollers. They introduce the basic WSim features listed above.

After a brief presentation of the tools, four simple examples will be discussed: an example involving LEDs, another example on timers and LPMs, an example demonstrating serial port communication and finally an example involving the whole Worldsens framework (including WSNet).

Chapter 2

Brief Presentation of the Tools

Following is a list of tools needed to carry out this tutorial:

- `mspgcc`: The GCC toolchain for the Texas Instruments MSP430 MCUs.
- `gdb` and `insight`: The GNU Project Debugger and its GUI.
- `wsim`: The Worldsens hardware platform simulator.
- `wconsole`: The Worldsens serial console.
- `wtracer`: The Worlsens trace reading tool.
- `wsnet`: The Worldsens wireless network simulator.

Before starting the tutorial, make sure you are using the Worldsens Live CD or that you have installed all the necessary tools listed above. The Worldsens suite will be described in the following. For information on the first couple of tools, please refer to their respective manuals. Please find below some useful links.

- The official mspgcc manual
- Courses and material on MSP430 programming
- The GDB documentation
- The GDB GUI FAQ

2.1 WSim: the Worldsens hardware platform simulator

WSim is part of the Worldsens development environment. WSim is a full platform simulator that can run the target platform object code without modification. The primary goals of WSim are debugging, profiling and performance evaluation for deeply embedded wireless sensor node/network (WSN) distributed applications.

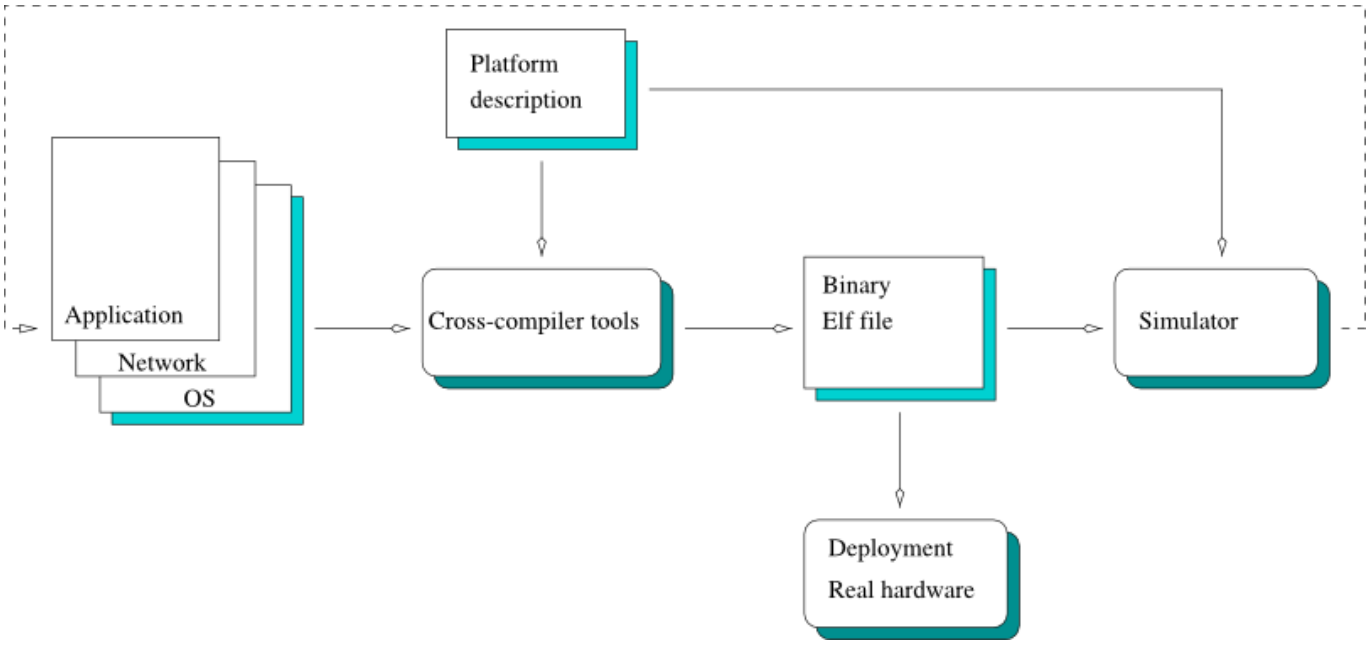


Figure 2.1: The place of WSim in the development process

Writing distributed WSN applications using the Worldsens toolkit is far easier than using prototype boards due to the fast compile/debug/estimate development cycle. This performance estimation is made possible using a full platform simulation at a cycle accurate level. Peripherals that deal with real time constraints (timers, interrupts, serial ports, radio interface, etc.) are included in the WSim platform model. Time is given in a cycle accurate manner using application instruction timing.

WSim provides a simulation environment in which you can get:

- Full instruction set support for the target microcontroller
- Cycle-accurate simulation
- Simulation of peripherals such as timers
- Interrupts
- Cycle-accurate simulation of other peripherals (e.g. UART)
- External peripherals (radio modules, LCD display, ...)
- A full system debug and performance analysis framework

WSim should compile and run on any Un*x-like platform. It has been tested on Linux/x86, Linux/x86_64, Linux/ppc32, Linux/Alpha, Solaris9/Sparc, Solaris10/Sparc, MacOSX/ppc32, MacOSX/x86 and Windows XP (using cygwin).

2.2 WConsole: the Worldsens serial console

The WorldSens serial console (WConsole) is a standalone application that can be used to drive the communication with the pseudo tty model. Its main use is to provide a full duplex interface that can output the octet stream from the simulator and use the host keyboard to communicate with the simulated application. The source code of the serial console can also be used to derive and test applications that will communicate through the serial port of the node in production use such as base station monitoring applications.

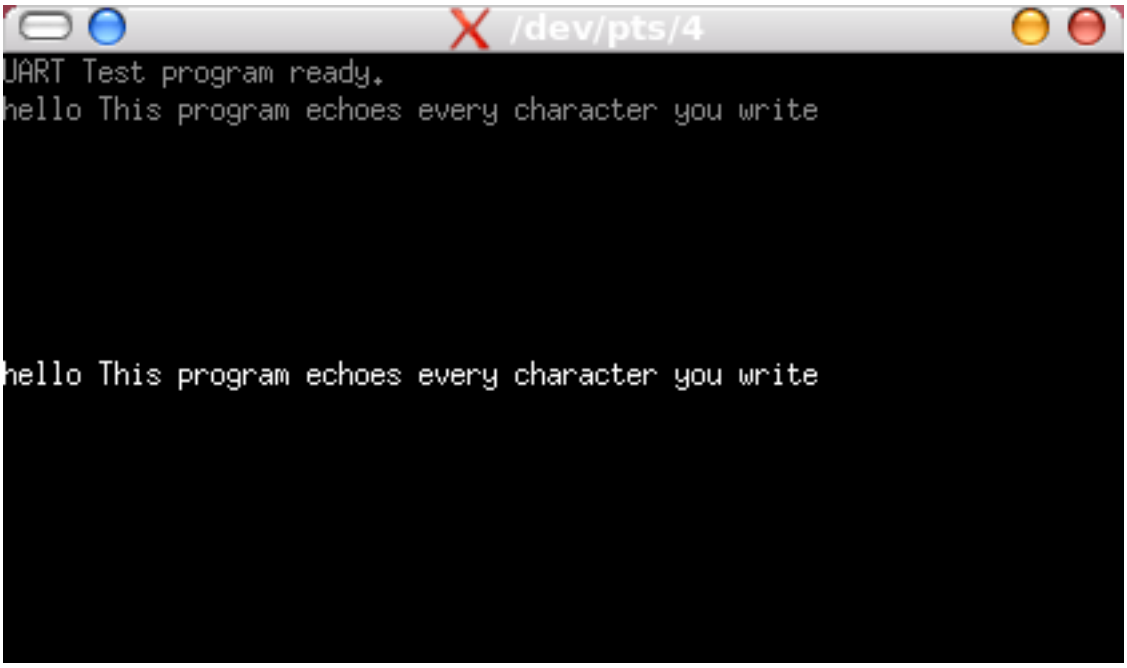


Figure 2.2: WConsole example

The screenshot of the WSim serial console shown in Figure 2.2 presents a simple application that produces an echo for every character sent to its serial port. The top half of the user interface window presents the simulation output with grayed letters and bottom half is the local keyboard input for the host machine.

2.3 WTracer: the Worldsens trace reading tool

WSim traces are output in binary format. The WTracer tool transforms the traces in the Gnuplot format.

2.4 WSNet: the Worldsens wireless network simulator

WSNet is part of the Worldsens development environment. WSNet is a modular event-driven wireless network simulator. Its architecture consists in different blocks that implement functionnalities of the nodes and the radio medium. As an example, separate blocks are defined for the radio propagation, the interference computation or the radio modulation. Similarly, a node is itself composed by several blocks: one for the mobility, one for the application, one for the system queue, one for the mac layer, one for the radio and one for the antenna.

During one simulation, the behaviour of a block is specified using a model. A model is one particular implementation of the block functionnalities. If we consider the radio propagation block, one model implements the free space propagation model; another one implements a radio range model, etc. Considering the radio modulation block, one model implements the BPSK modulation, another one the FSK modulation, etc.

Models are either provided with the simulator or developped by users. A user can develop models for any of the blocks and use it for its own simulations. A model must implement a given API as well as characteristics that are defined by its associated block. WSNet should be able to compile on any Unix-like platform and has been tested on Linux/x86, Linux/x86 64, Linux/ppc32, Linux/Alpha, Solaris9/sparc, Solaris10/sparc, MacOSX/ppc32, Windows XP (using cygwin).

2.5 Worldsens: the development framework

WSim can be attached to the WSNet simulator through radio interfaces such as the Chipcon CC1100 model. This allows to build a complete simulation framework that is able to debug and analyse a complete wireless sensor network. Thus WSim and WSNet allow to design and refine a full distributed application from the top level specifications to the final application binary within the same framework and application description: the real application code.

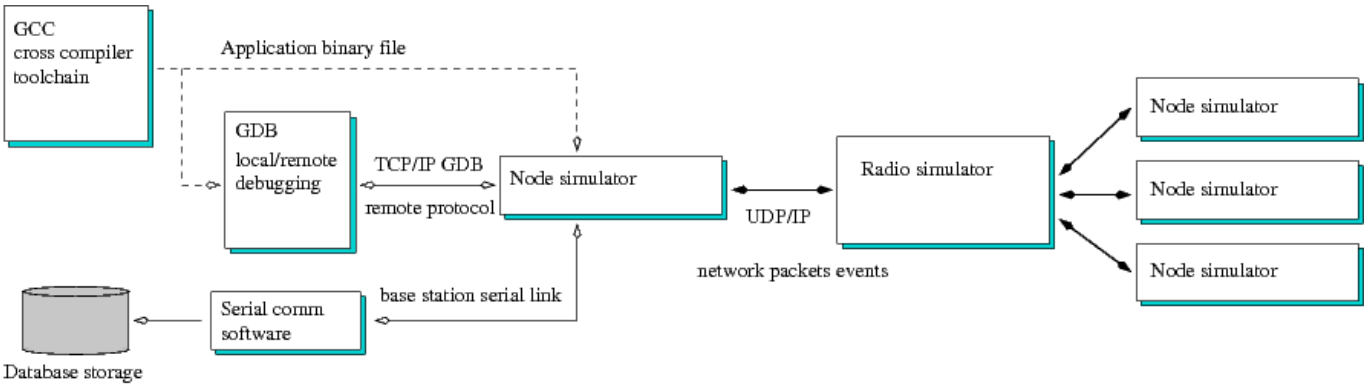


Figure 2.3: Distributed simulation

Communication between the simulators is done using IPv4 unicast and multicast in order to allow the distribution of the simulation load across a network of workstations. A typical middle range workstation can handle several dozens of concurrent WSN node simulators so that a small set of workstations can scale to hundreds of nodes. This allows the Worldsens environment to provide a performance analysis of real world applications using cycle accurate simulation tools.

These features will be demonstrated in Chapter 6.

Chapter 3

LEDs Example

3.1 Introduction

This chapter presents the LEDs (Light Emitting Diodes) example. It is a very simple application written for the WSN430 hardware platform, in which LEDs are turned on and off at variable time intervals. The purpose of this example is to show you how to *execute* a program using WSim, how to *convert the traces* generated by the simulator into a human readable format and how to *debug* your program using Insight/GDB.

3.2 Compiling and executing the example

If you are using the Worldsens Live CD, go to `/opt/worldsens/examples/wsn430-leds`. If not, go to `$WSIM_FOLDER/exa` and type:

```
[wsn430-leds]$ make
```

You should see:

```
msp430-gcc -mmcu=msp430x1611 -O2 -Wall -g   -c -o main.o main.c
msp430-gcc -mmcu=msp430x1611 -o wsn430-leds.elf main.o
msp430-objcopy -O ihex wsn430-leds.elf wsn430-leds.a43
msp430-objdump -dSt wsn430-leds.elf >wsn430-leds.lst
```

If you are not using the Worldsens Live CD and you see an error, make sure all the tools are in your `$PATH`.

Once the example is compiled, you can execute it using WSim with the following options:

- `ui`: to show the user interface (i.e. the LEDs blinking).
- `trace`: write simulator traces in a file (`wsim.trc`).
- `mode`: WSim mode (`time`, `steps` or `gdb`); if this is not specified, WSim will run forever.
- `modearg`: if mode is `time` or `steps`, then this option will specify the simulation time in nanoseconds or number of steps.

The command to type is therefore the following (expected output is included below):

```
[wsn430-leds]$ wsim-wsn430 --ui --trace --mode=time --modearg=100000000000 wsn430-leds.elf
WSim 0.82cvs, copyright 2005, 2006, 2007 Worldsens
wsim:pid:20530
wsim:log:wsim.log
```

The mode option
As explained above, when the `mode` option is not used, simulation time is unlimited. It is important to note that, in this case, trace data are kept in a buffer of relatively large size (approx. 4 MB). Therefore, should the user interrupt the simulation (`killall -USR1 wsim-wsn430`), they should expect the trace files to be empty.

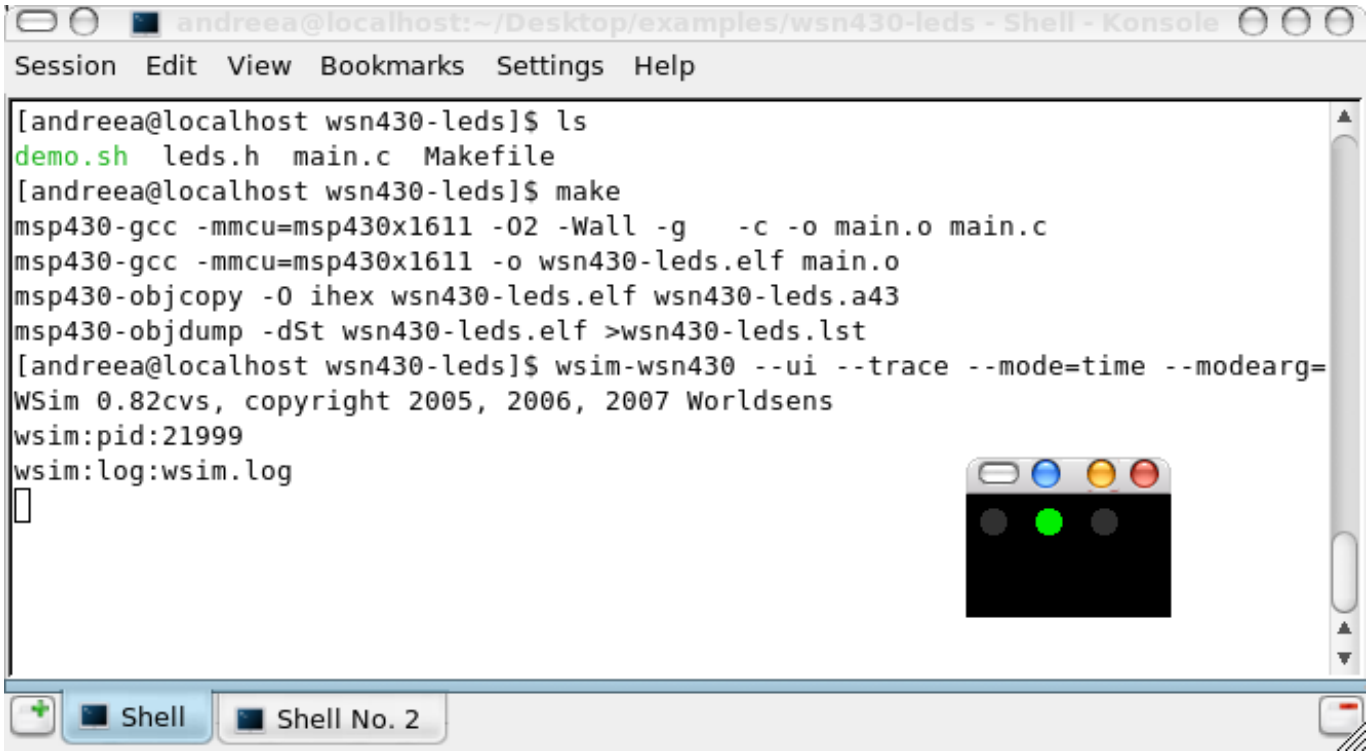


Figure 3.1: Snapshot of the execution of the LEDs example

3.3 Converting and interpreting the traces

Simulation traces are written in a file called `wsim.trc`, if the `trace` option was used. The trace file records interruptions, blinking frequency for each of the three LEDs, clock frequencies, power mode changes, serial port information etc. However, this information is not readily available, since it is recorded in binary format. This is why it first needs to be converted to a human readable form. To do so, we will use a tool called WTracer, which is part of the Worldsens tool suite. After having executed the LEDs example, type the following:

```
[wsn430-leds]$ wtracer --in=wsim.trc --out=wsim.gp --format=gplot
```

You should now have a new file, `wsim.gp` in Gnuplot format. The Gnuplot utility will use this file to generate nice plots of the recorded information (one EPS file per type of data). To do so, type the command:

```
[wsn430-leds]$ cat wsim.gp | gnuplot
```

The files resulting from this procedure are: `wsim.gp.CC1100_state.eps`, `wsim.gp.ds2411.eps`, `wsim.gp.Interrupt.e`
`wsim.gp.led1.eps`, `wsim.gp.led2.eps`, `wsim.gp.led3.eps`, `wsim.gp.m25p80.eps`, `wsim.gp.msp430_aclk.e`
`wsim.gp.msp430_mclk.eps`, `wsim.gp.msp430_smclk.eps`, `wsim.gp.Power_mode.eps`, `wsim.gp.Usart0.eps`,
`wsim.gp.Usart1.eps`. However, since this simple example only makes use of LEDs, the LED files are the only ones relevant. They show the LED blinking frequency during the entire simultation time. The file for LED 1 is included below, in Figure 3.2.

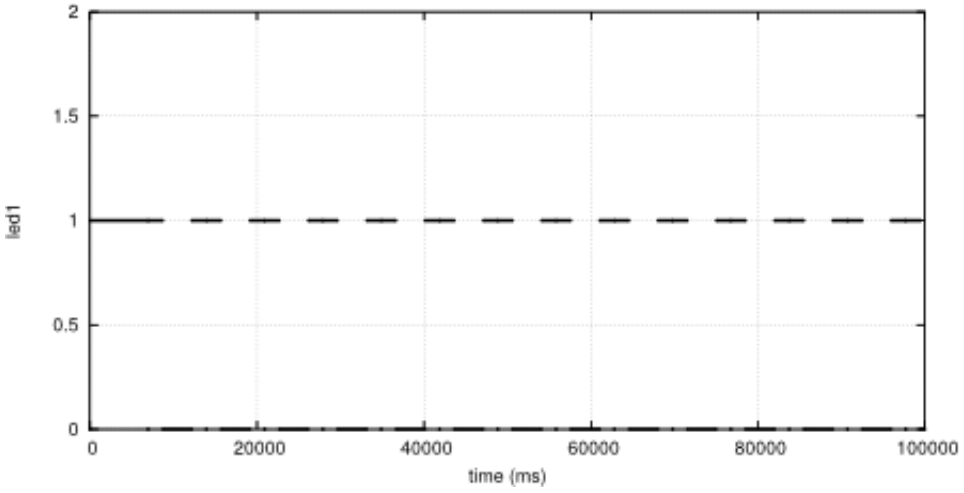


Figure 3.2: Trace of LED 1

3.4 Debugging using WSim and Insight/GDB

WSim understands the GDB remote protocol to drive the simulation engine. The simulator can be started using the `--mode=gdb` parameter to use this functionality. The following steps should be followed in order to debug an application using WSim and Insight/GDB:

1. Start a simulation in debug mode. WSim will stop right after machine creation and will wait on the default port (TCP 2159) until a connexion is made from a remote debugger that implements the same target architecture (GDB must be compiled as a cross-debugger for the target architecture):

```
[wsn430-leds]$ wsim-wsn430 --ui --trace --mode=gdb wsn430-leds.elf
```

2. Launch Insight/GDB and load the ELF file as shown below (File menu --> Open):

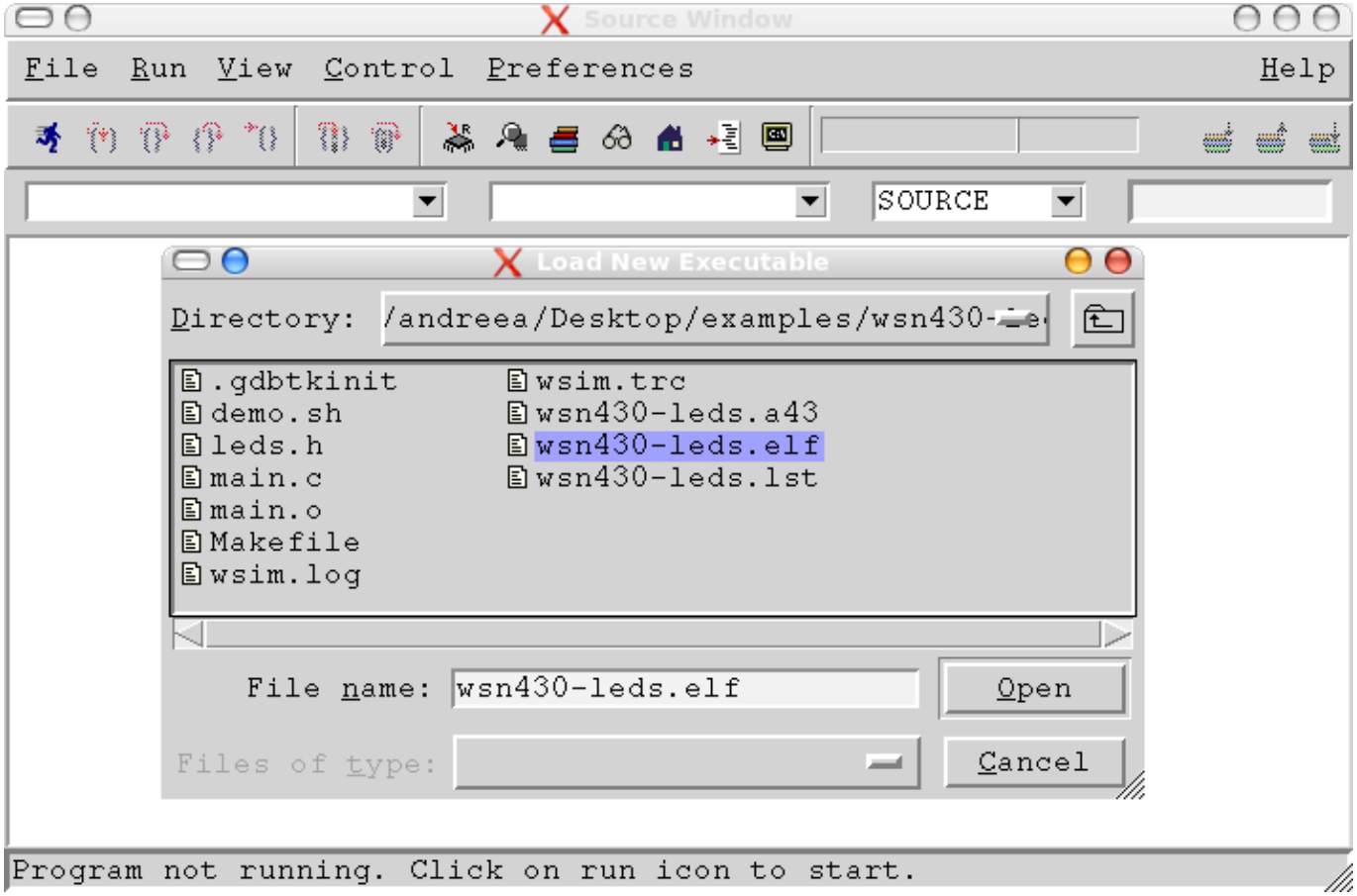


Figure 3.3: First debugging snapshot

3. Connect to the WSim simulation as shown below (Run menu --> Connect to target):

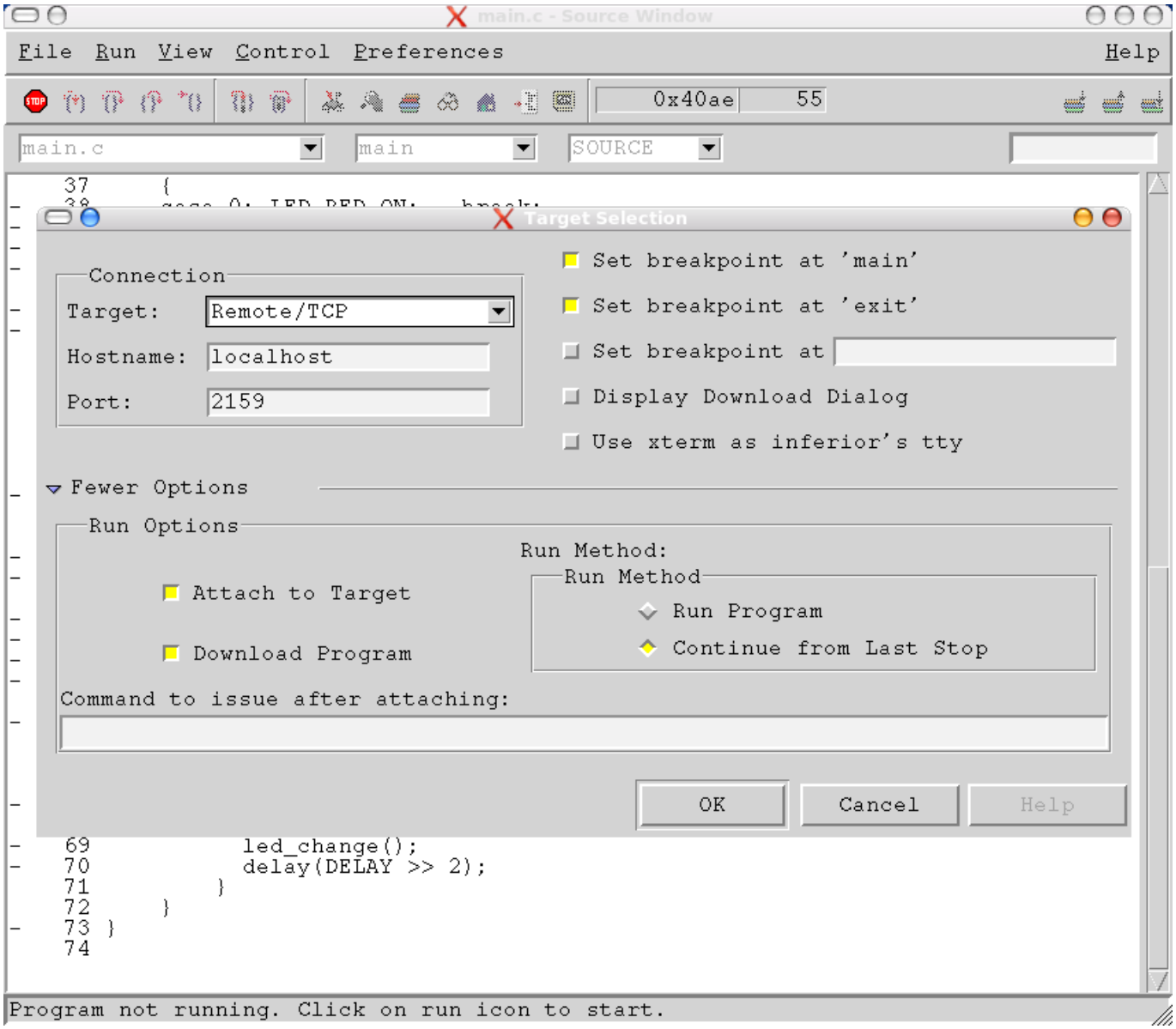


Figure 3.4: Second debugging snapshot

Once you click OK, you should see the following:

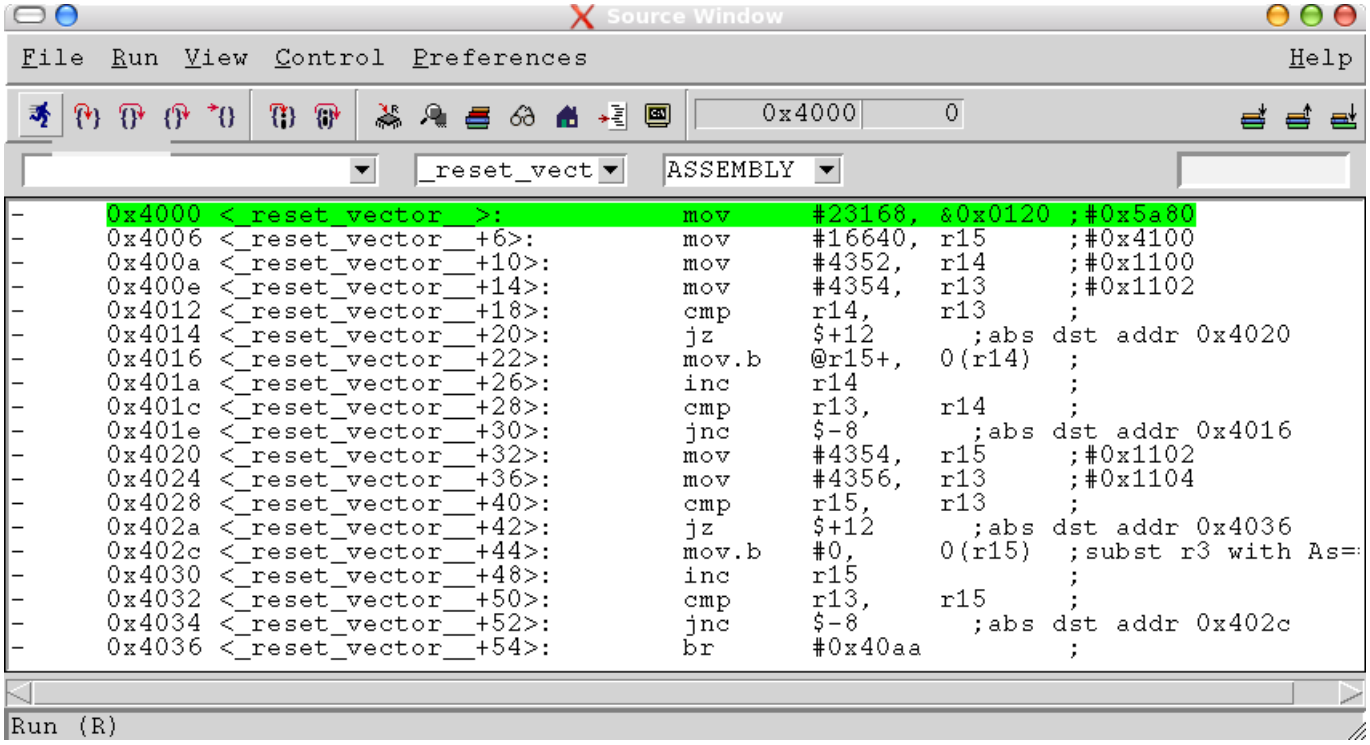


Figure 3.5: Third debugging snapshot

4. Start the execution by pressing the « Run » button (upper left corner). Don't worry about breakpoints, two of them are automatically set for you: one in the beginning of the main function and a second one at the end of the function. You should see this:

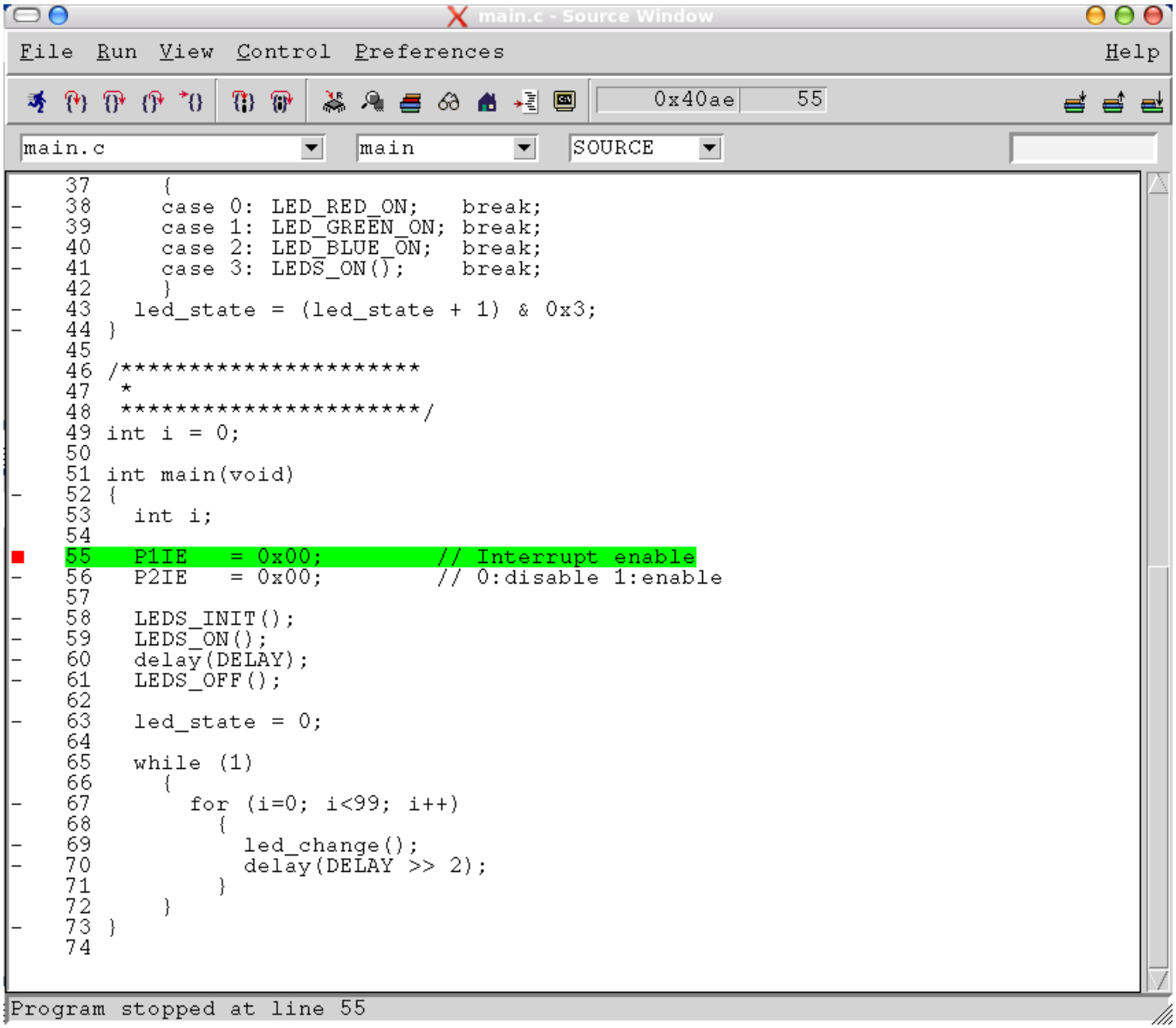


Figure 3.6: Fourth debugging snapshot

5. Now you can start setting your breakpoints and debugging!

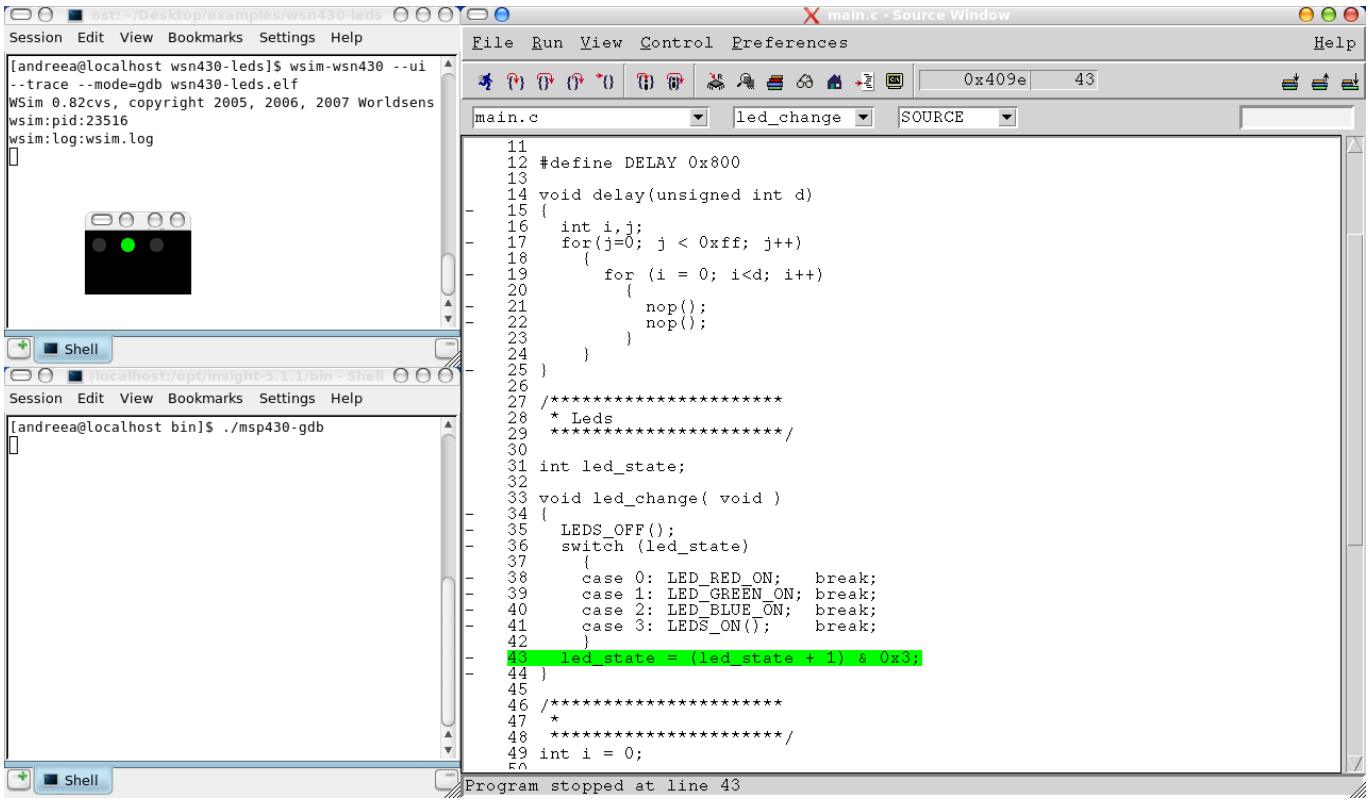


Figure 3.7: Fifth debugging snapshot

6. Stop debugging and close the file: File menu --> Close.

Chapter 4

Timer Example

4.1 Introduction

This chapter presents the timers example. This application written for the WSN430 hardware platform aims at demonstrating WSim's capability to simulate interrupts. We will also discuss setting breakpoints in an application using LPMs (Low Power Modes).

4.2 Compiling and executing the example

If you are using the Worldsens Live CD, go to `/opt/worldsens/examples/wsn430-timer`. If not, go to `$WSIM_FOLDER/ex` and type:

```
[wsn430-timer]$ make
```

You should see:

```
msp430-gcc -mmcu=msp430x1611 -O2 -Wall -I../wsn430-leds -I../wsn430-clock -I../wsn430- ↩
    serial -g   -c -o main.o main.c
msp430-gcc -c -mmcu=msp430x1611 -O2 -Wall -I../wsn430-leds -I../wsn430-clock -I../wsn430- ↩
    serial -g ../wsn430-clock/clock.c
msp430-gcc -c -mmcu=msp430x1611 -O2 -Wall -I../wsn430-leds -I../wsn430-clock -I../wsn430- ↩
    serial -g ../wsn430-serial/uart1.c
msp430-gcc -mmcu=msp430x1611 -O2 -Wall -I../wsn430-leds -I../wsn430-clock -I../wsn430- ↩
    serial -g   -c -o timer.o timer.c
msp430-gcc -mmcu=msp430x1611 -o wsn430-timer.elf main.o clock.o uart1.o timer.o
msp430-objcopy -O ihex wsn430-timer.elf wsn430-timer.a43
msp430-objdump -dSt wsn430-timer.elf >wsn430-timer.lst
```

If you are not using the Worldsens Live CD and you see an error, make sure all the tools are in your `$PATH`.

Once the example is compiled, you can execute it using WSim with the same options presented in the previous example, i.e.:

- `ui`: to show the user interface (i.e. the LEDs blinking).
- `trace`: write simulator traces in a file (`wsim.trc`).
- `mode`: WSim mode (`time`, `steps` or `gdb`); if this is not specified, WSim will run forever.
- `modearg`: if mode is `time` or `steps`, then this option will specify the simulation time in nanoseconds or number of steps.

The command to type is therefore the following (expected output is included below):

```
[wsn430-timer]$ wsim-wsn430 --ui --trace --mode=time --modearg=100000000000 wsn430-timer. ↩
    elf
WSim 0.82cvs, copyright 2005, 2006, 2007 Worldsens
wsim:pid:22830
wsim:log:wsim.log
```

The mode option
As explained above, when the `mode` option is not used, simulation time is unlimited. It is important to note that, in this case, trace data are kept in a buffer of relatively large size (approx. 4 MB). Therefore, should the user interrupt the simulation (`killall -USR1 wsim-wsn430`), they should expect the trace files to be empty.

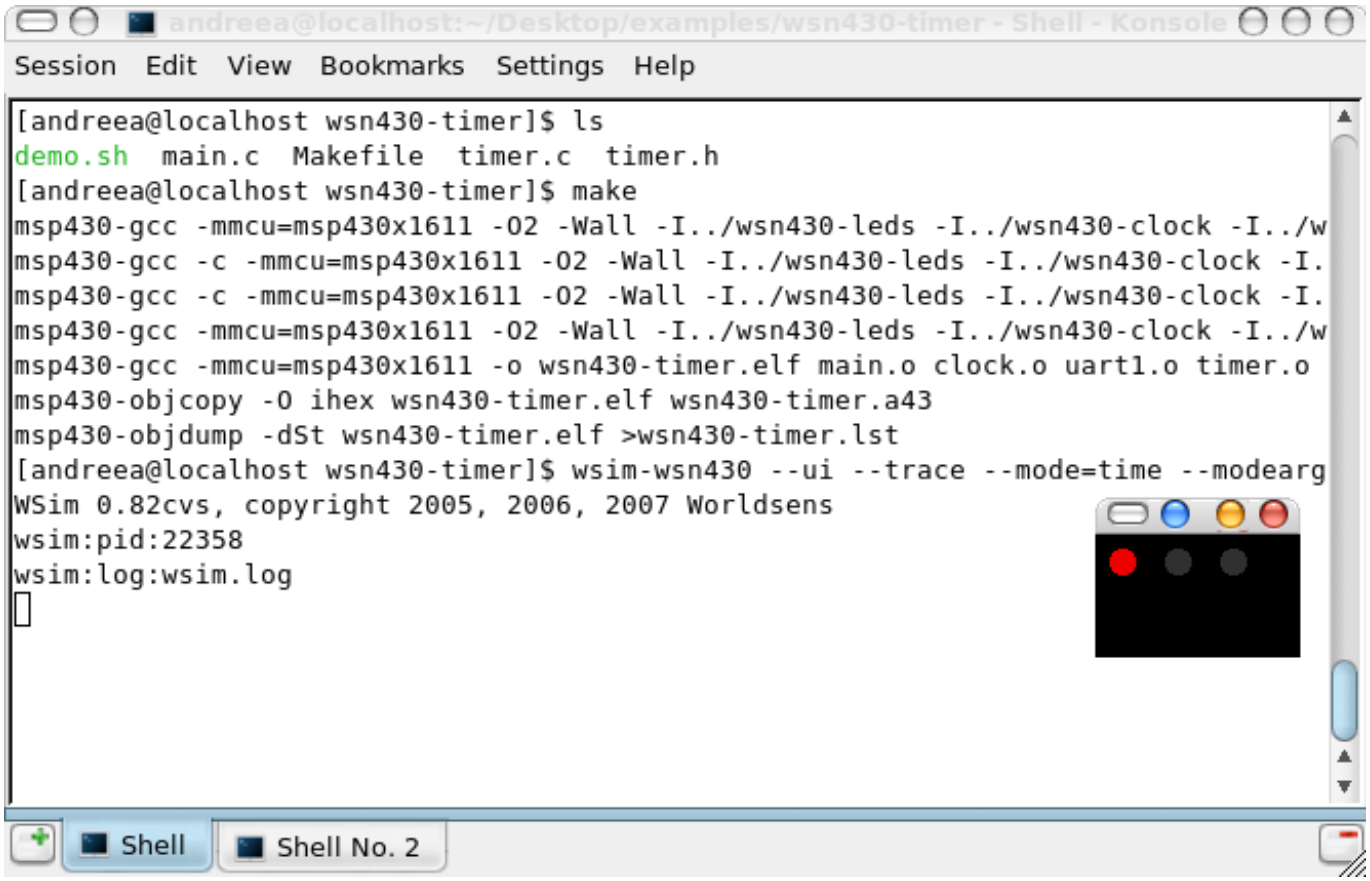


Figure 4.1: Snapshot of the execution of the timer example

4.3 Converting and interpreting the traces

After having converted the trace file as explained in the previous example, you will again obtain EPS files for each type of data: `wsim.gp.CC1100_state.eps`, `wsim.gp.ds2411.eps`, `wsim.gp.Interrupt.eps`, `wsim.gp.led1.eps`, `wsim.gp.` `wsim.gp.led3.eps`, `wsim.gp.m25p80.eps`, `wsim.gp.msp430_aclk.eps`, `wsim.gp.msp430_mclk.eps`, `wsim.gp` `wsim.gp.Power_mode.eps`, `wsim.gp.Usart0.eps`, `wsim.gp.Usart1.eps`. However, this time, besides from LED files, another interesting file to analyse is the interrupt file, `wsim.gp.Interrupt.eps`, which presents harware interrupts issued during the execution of the program by the simulator. This file is included below, as well as LED trace files:

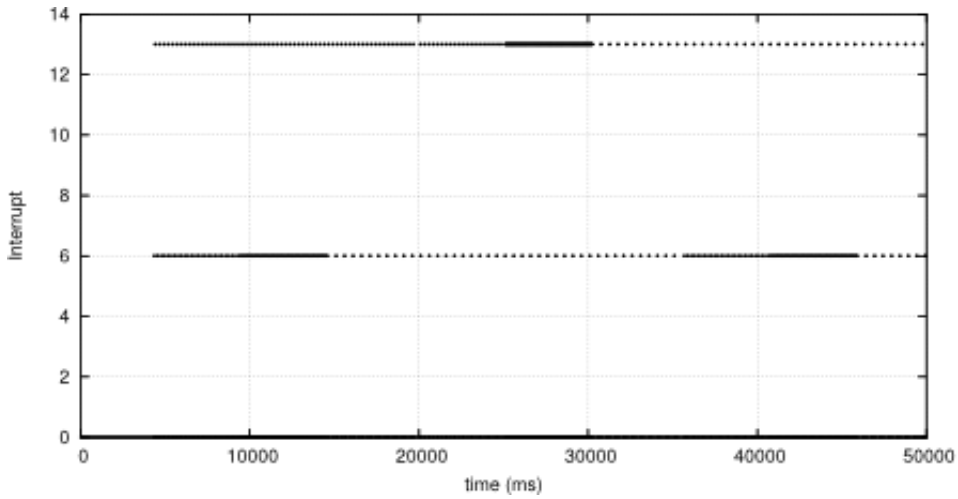
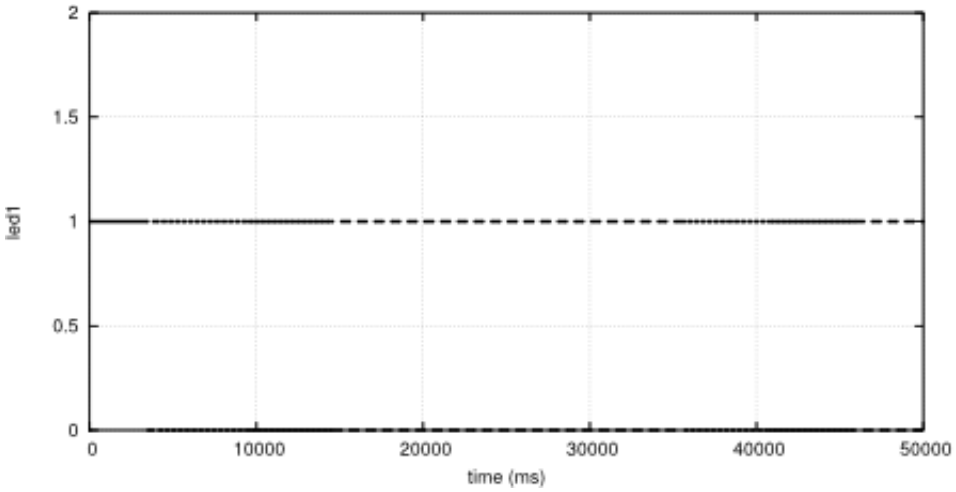
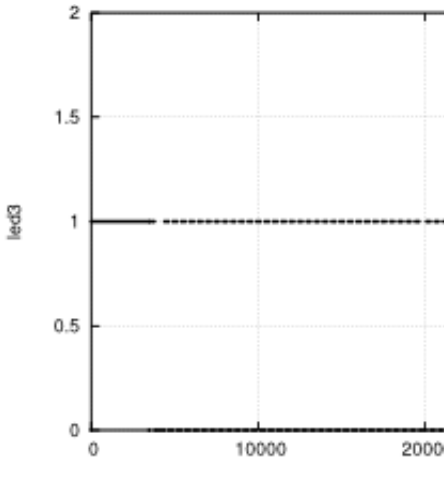


Figure 4.2: Interruptions for the timer example





Chapter 5

Serial Port Example

5.1 Introduction

This chapter presents the serial port example. Like the previous examples, this one is also written for the WSN430 hardware platform and it demonstrates communication through the serial port in WSim. It is a simple application that produces an echo for every character sent to its serial port 1.

5.2 Compiling and executing the example

To compile the example, follow the same instructions as before. For users of the Worldsens Live CD, the folder containing this example is `/opt/worldsens/examples/wsn430-serial`. For other, go to `$WSIM_FOLDER/examples/wsn430-serial`.

To execute the example users will have to follow the procedure below:

1. Launch an instance of the Worldsens WConsole application, as follows:

   ```
   [wsn430-serial]$ wconsole
   ```

   You should see this:

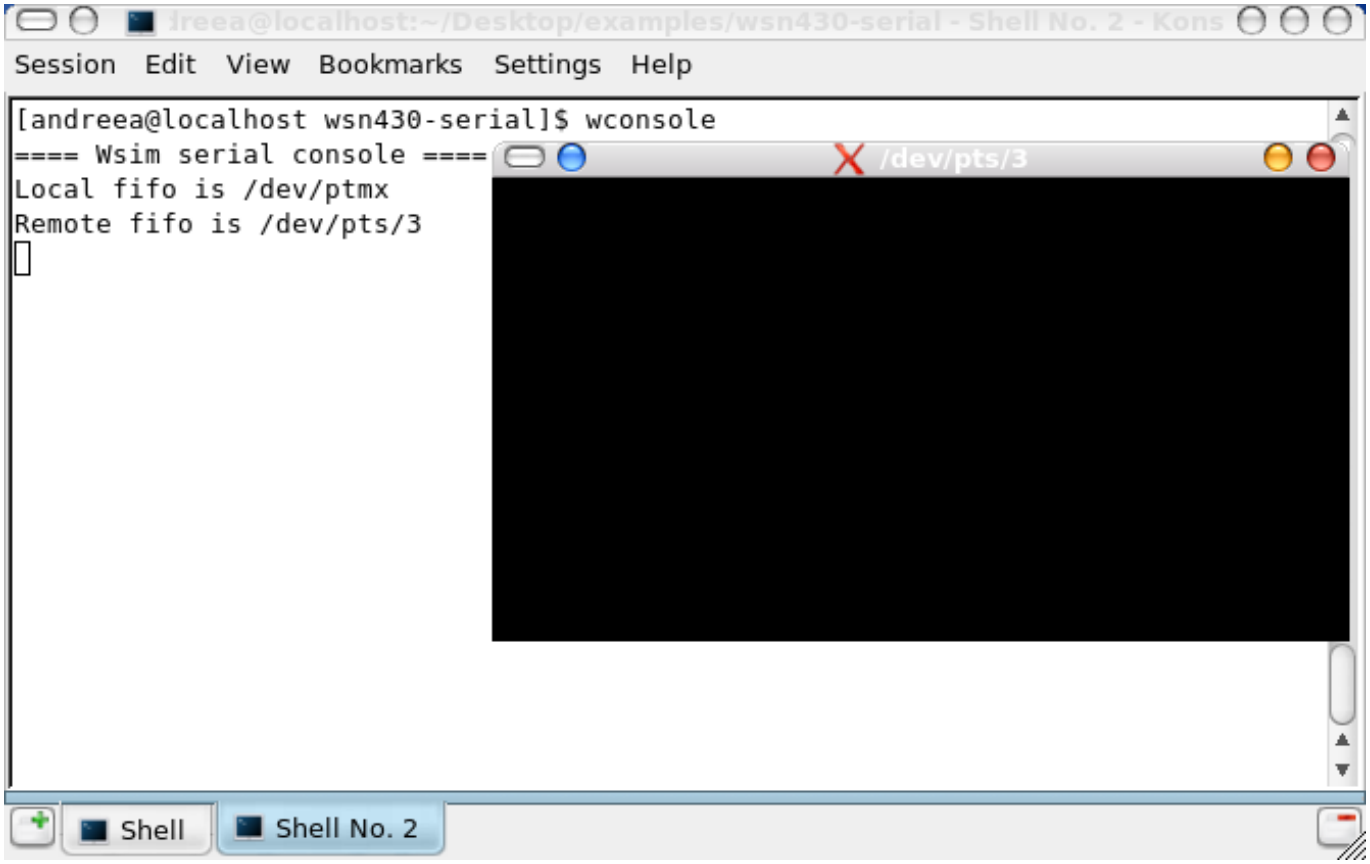


Figure 5.1: Snapshot of the launching of the serial console

2. Launch the simulator and connect it to the previously launched WConsole instance, using the following command (expected output is included):

```
[wsn430-timer]$ wsim-wsn430 --serial=/dev/pts/3 wsn430-serial.elf
WSim 0.82cvs, copyright 2005, 2006, 2007 Worldsens
wsim:pid:4343
wsim:log:wsim.log
```

Once you attach the serial console to the simulated platform's serial port 1, the application will let you switch between its two modes by typing « Z ». The two available modes are:

- IRQ+LPM0: in this mode the program starts by checking if a character is available on the serial port; if no character is available, it goes into a Low Power Mode (see the TI MSP430 User Guide), LPM0 to be precise; all subsequent characters are processed in interrupt routines.
- polling: in this mode, the program is constantly checking if the user has entered any new character on the serial port.

The top half of the user interface window presents the simulation output with grayed letters and bottom half is the local keyboard echo for the host machine. In both modes, all the text you type will be echoed by the application in the upper part of the serial console.

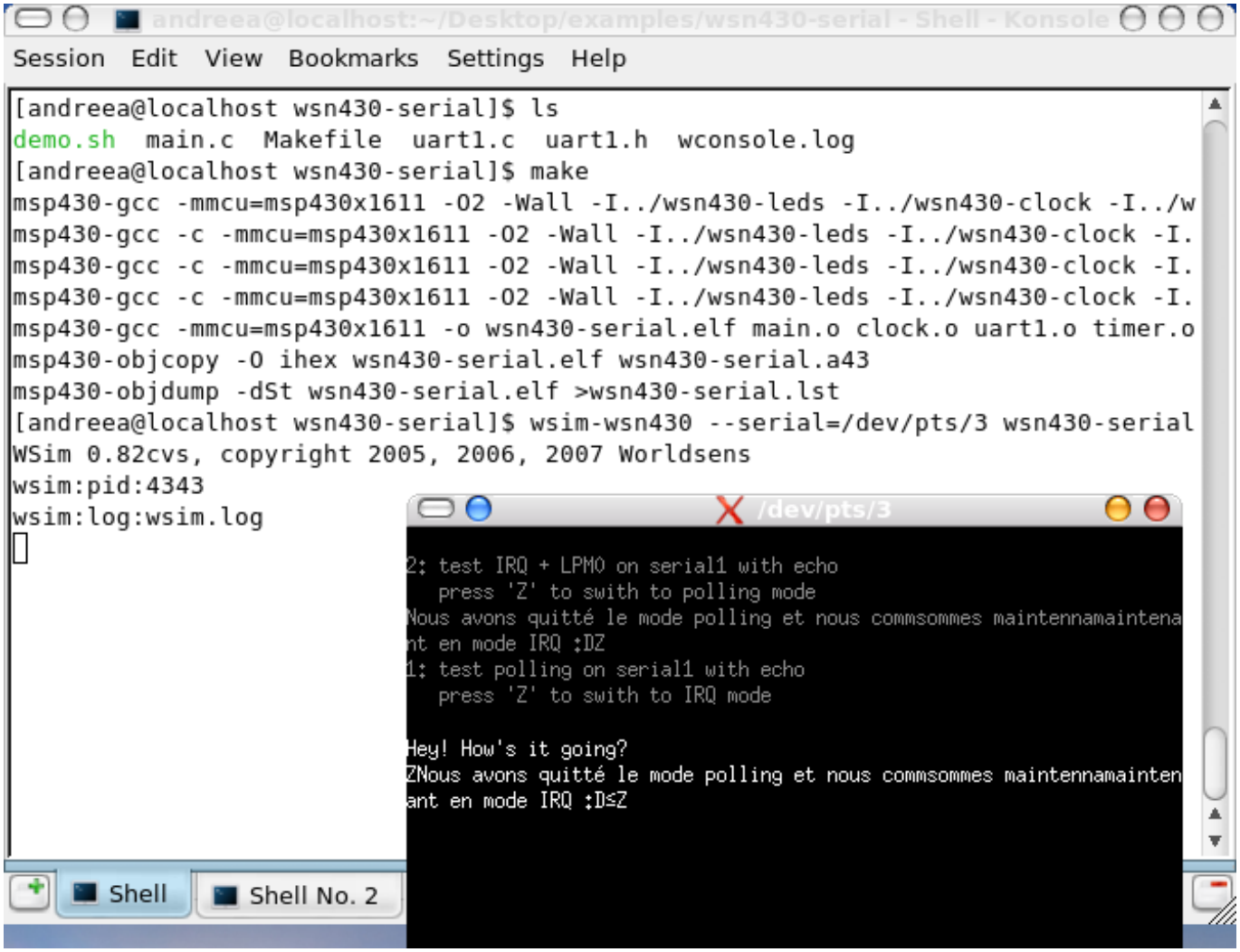


Figure 5.2: Snapshot of the execution of the serial example

5.3 Debugging and LPMs

As explained above, the IRQ+LPM0 mode consists of a sequence of LPM0 periods and executions of interrupt routines. For this reason, bla bla bla.

Chapter 6

Token Passing Example

6.1 Introduction

This example shows how to start a mixed simulation using WSim for node simulation and WSNet for radio propagation and interferences. It consists of three sensor nodes (again, simulating the WSN430 hardware platform). The nodes are communicating with one another by passing a token around the network.

6.2 Example description and execution

Nodes are numbered from 1 to 3 and can identify themselves through the DS2411 unique Id number chip. Node Id are given at simulation startup using the `--ds2411=0a:00:00:00:00:00:01:01`. Node Ids must be given with the right CRC value (a valid CRC is calculated at simulation startup). The `--node-id` parameter is used by WSim for packet exchanges with WSNet. This node Id cannot be used by the embedded software running on the node hardware. The demo application is a single token passing application. Each node listens the radio medium and waits for a packet that contains its node-Id. Once a packet is received with the correct id number, the corresponding node *bla bla bla.*

To compile this example, simply run the `make` command. To execute it, the following procedure must be followed:

1. Launch three WConsole instances (each node will print its feedback in these serial consoles) in separate shells.
2. Launch WSNet the network simulator. You must either launch it from the directory containing the example or provide the path of this directory to it as an option (see WSNet Tutorial for more details). This is needed for the configuration of the network simulator (environment, physical layer, MAC layer etc).
3. Launch the three WSim nodes. To do so, use the script provided with the example: `demo.sh`. For regular execution (time mode, 10 seconds), use the script with the following arguments:

```
[wsn430-token]$ ./demo.sh p3 /dev/pts/7 /dev/pts/8 /dev/pts/9
```

Figure 6.1 shows the console output of the 3 nodes. Console duplicated messages are one of the effects of the distributed simulation mechanims. WSNet keeps track of global simulation time but does not impose a tight synchronisation with WSim instances. Time synchronisation is done among nodes using an optimistic behaviour that lets processes work at full speed, only requiring time synchronisation when uncoherent events are detected. Correct distributed events ordering is ensured by backtracking nodes that are in advance compared to others. Only WSim processes are backtracked but WConsoles do not go back in time, hence the duplicated messages on consoles output.

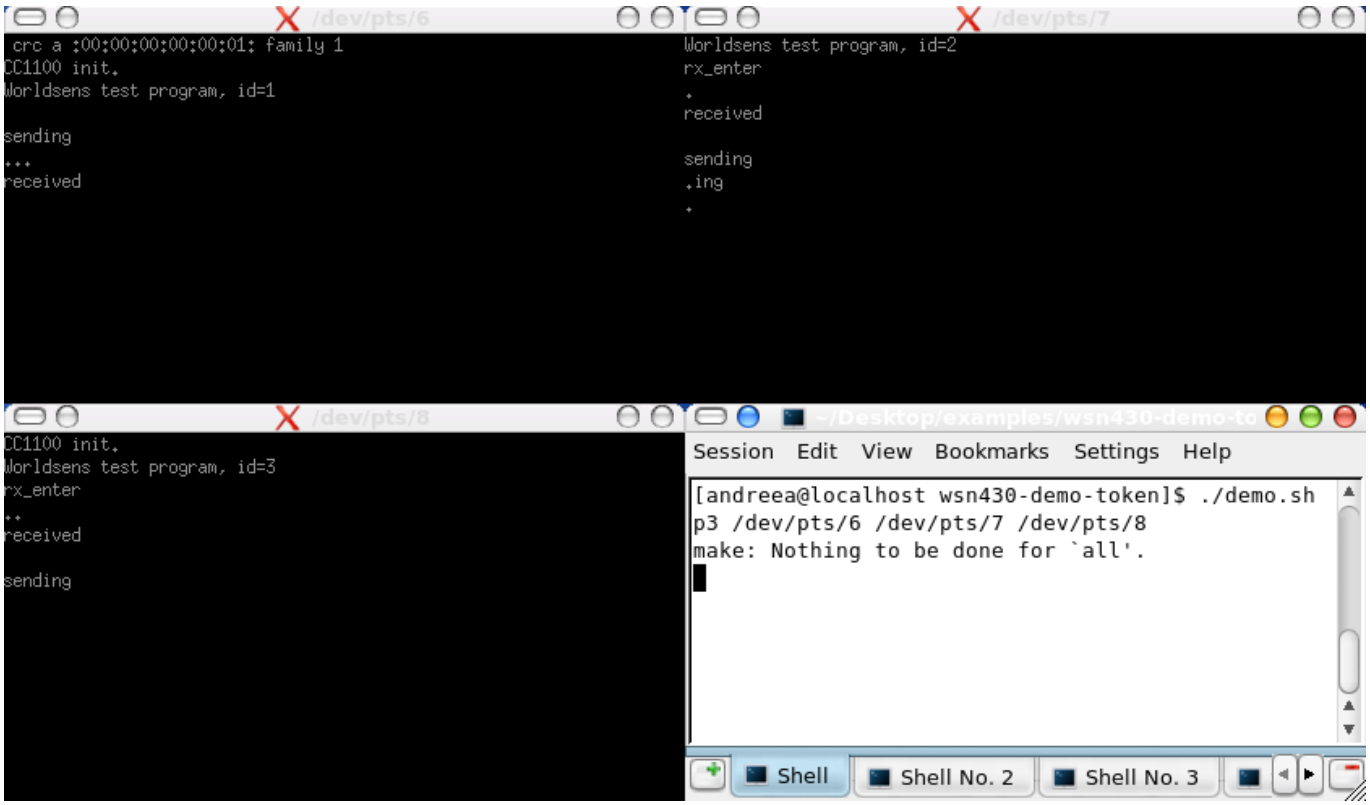


Figure 6.1: Snapshot of the execution of the token passing example

6.3 Debugging a distributed application

WSim allows debugging even for distributed applications. It is possible to place one node from the network in debug mode and the entire application (including other nodes and WSNet) will follow the node being debugged.

To launch this example in debug mode, follow the same steps as above, with a slight modification at the third step:

- To place Node 1 in debug mode and the other two in regular execution, execute the script with the following arguments:

```
[wsn430-token]$ ./demo.sh d31 /dev/pts/7 /dev/pts/8 /dev/pts/9
```

- To place Node 2 in debug mode and the other two in regular execution, execute the script with the following arguments:

```
[wsn430-token]$ ./demo.sh d32 /dev/pts/7 /dev/pts/8 /dev/pts/9
```

Once the simulators are launched, you can connect GDB/Insight to the node placed in debug mode in the same way as presented in Chapter 3.

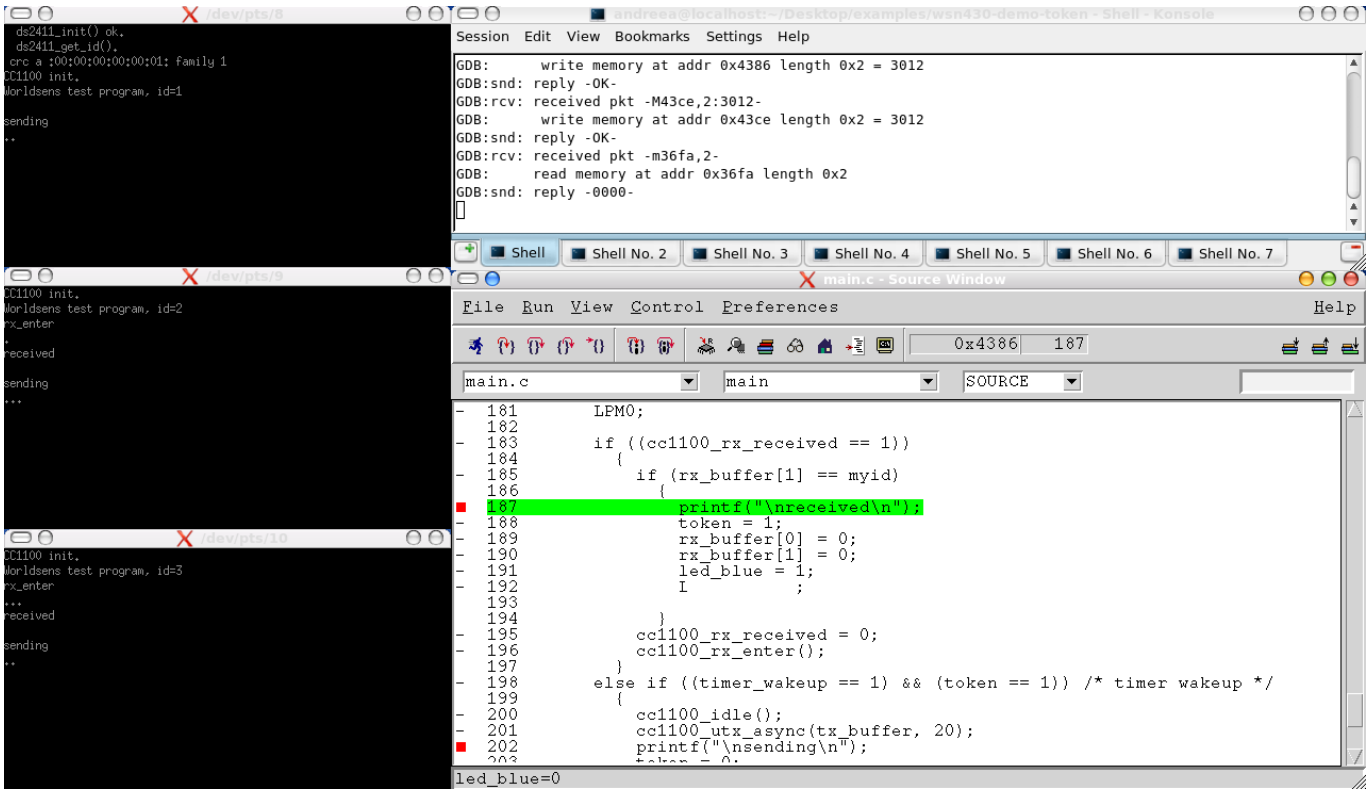


Figure 6.2: Snapshot of the debugging of the token passing example

Figure 6.2 gives an idea of how the debugging of this example should look like.

Chapter 7

Credits

Credit for the icons goes to the Tango Desktop Project.

The style is inspired from Shaun's Blog.