Virtual machine for MSP430 development

The virtual machine[1] is an Ubuntu based minimal distribution, extended with MSP430 related development tools. The VM is based on a VM that was developed within the European WASP project, so there might be references to that project, for example in the form of user names and directories. To run the virtual machine, you need to install VMware[2] or use the local copy of version 3.1.3-324285[3]. The **username** and **password** for the main account are 'wasp' and 'wasp'.

Course material

Documentation for the practical training can be found as documents of type "Practical training" in OASE:

- assignments: `PWx.pdf`;
- source code: `2IMN20-Practical-sources-weekx.zip`

Getting started

After retrieving the latest version of the source code, the typical work flow is

- *Modify the program*
  Several options are available for editing the source code. The virtual machine contains a number of text editors (*vi*, *GVIM* and *nano*). It is also possible to use an editor from the host operating system by mounting the home directory as a network disk. The instructions for mounting the directory are provided when you open a new terminal window.

- *Compile the program*

  ```
  $ make
  ```

  The command `make` uses the file `Makefile` to convert the source code into a binary ELF file, based on modifications that were made to the individual files. It is possible to add addition rules to that file, such that other output is generated as well. You may find it interesting to take a look at some of the files produced by make. You can disassemble compiled (`.o`) files with, e.g., `msp430-objdump -D -S SchedulerNPBasic.o`. The `-D` option causes all sections to be disassembled and the `-S` option produces source statements mixed with the disassembly.

[1] `http://www.win.tue.nl/san/education/2IN26/Course-2013-Ubuntu-spin3.7z`
[2] `http://www.vmware.com/products/player/`
[3] `http://www.win.tue.nl/san/education/2IN26/VMware-player-3.1.3-324285.exe`

- *Generate a trace file*

  ```
  $ wsim-iclbsn2 --ui --trace=sched.trc --mode=time --modearg=5s SchedTest.elf
  ```

  The command `wsim-iclbsn2` simulates an MSP430-based hardware platform and runs the ELF file `SchedTest.elf`. The selected `time` mode ensures that the simulation is stopped after a certain time interval, in this case `5` seconds. A trace of the simulation is saved into file `sched.trc`. The `--ui` option indicates that a graphical user interface is shown during the simulation, indicating the activity of the LEDs.

- *Convert the format of the trace file*

  ```
  $ wtracer --in=sched.trc --out=wsim.vcd --format=vcd
  ```

  The command `wtracer` converts the trace file `sched.trc` into the trace file `wsim.vcd`, using the format `vcd`.

- Visualize the result

  ```
  $ gtkwave wsim.vcd
  ```

  The command `gtkwave` opens a graphical user interface to inspect the traces from the input file, in this case `wsim.vcd`. In the SST and Signals sections, you can select the variables, registers and LEDs that were traced during the simulation.

Additional information

The following information is not required for finishing the practical part of the course. However, when you want to investigate the behaviour of the code more thoroughly, some hints are provided on how to use the tools mentioned above for more advanced analysis.

GCC MSP430 compiler

Two versions of the MSP430 compiler are provided, version 3.3.6 and version 4.4.2, which are located in `/opt/msp430-gcc-3.3.6` and `/opt/msp430-gcc-4.4.2`, respectively. The default version is 4.4.2. The compiler tool chain includes the cross compiler, the debugger, the standard C libraries and several tools to manipulate or inspect ELF files for the MSP430. In case you already have a virtual machine with Linux, you could try to install the MSP430 *gcc* versions from a separate archive[4]. Additional information about the MSP430 processor is available in the document MSP430 - general.pdf that is available in OASE.

[4]`http://www.win.tue.nl/san/education/2IN26/msp430-gcc-versions.tar.gz`

WSIM - Instruction level simulator

The instruction level simulator[5] is provided in /usr/local/bin. The simulator can be used to run MSP430 ELF files for different platforms. It is possible to run the simulator to observe the behavior of the LEDs (stand alone), to debug your code (in combination with *msp430-insight*) or to create trace files (which can be converted with *wtracer* and viewed with *gtkwave*).
The WSIM simulator tools indicate which command line options are available by calling it with the `--help` option:

```
wsim-iclbsn2 --help
```

The syntax for tracing specific variables or memory locations, is not provided in the output of this command. When you want to trace the **global** variable `counter`, the following command line simulates the code for one second and traces the value of `counter`.

```
wsim-iclbsn2 --trace=sched.trc --mode=time --modearg=1s --monitor=counter:w SchedTest.elf
```

If you want to trace several variables, you need to include the `:w` after each variable and separate the variables with a comma. In the WSIM tutorial[6] the `--monitor` option is explained further. Note that you have to add the `--trace` option as well.

If you leave out `--mode=time and --modearg=1s`, the simulator will run forever.

The simulator has different execution modes, which are specified with the option `--mode=`*mode*. The most useful modes are

- `gdb`

  For debugging your code and inspecting variables and registers. Do not include the `--modearg` in the command to start the simulator:

  ```
  wsim-iclbsn2 --ui --trace=sched.trc --mode=gdb SchedTest.elf
  ```

  When this mode is used, you have to start *msp430-insight* separately (in a different terminal, in the same directory). After starting it with

  ```
  msp430-insight --se=SchedTest.elf
  ```

  you can connect to the simulator by using the menu *Run – Connect to target* with the following settings:

  ```
  Target:  Remote/TCP
  Hostname:  localhost
  Port:  2159
  ```

[5] `http://wsim.gforge.inria.fr`
[6] `http://wsim.gforge.inria.fr/tutorials/monitor-option/monitor.html`

As mentioned above, the `--ui` argument of `wsim-iclbsn2` simulates the LEDS on the *WSIM* user interface. When simulating a run to be analyzed with *gtkwave*, this merely causes a slower execution of the simulator, but when using the simulator in debug mode (see below) it can be useful.

- `time`

  For running the execution for a specific duration, for example to create a trace file. You can specify the duration with `--modearg=15s` to get 15 seconds of simulation. This mode ensures that the generated trace file is correctly closed, such that it can be processed by *wtracer*. Note that the trace file can quickly become very large, especially when you trace many values that change rapidly (such as the program counter, the stack pointer or a full-speed LED toggle).

To analyse the exact control flow of the application, you can trace the program counter (PC), which is updated after every instruction and indicates which code is active. Due to the update frequency of the program counter, it is best to combine option `--msp430_trc_pc` (which enables PC tracing) with a short simulation time (`--mode=time --modeargs=1s`). In *gtkwave*, the program counter will be available in the msp430 signal section with the name `PC[15:0]`. It is explained below how you can visualize the name of the method that is executing at a certain moment in *gtkwave*.

Since the MSP430 processor is mainly used in embedded system, the user interface only simulates the LEDs to provide some feedback. Quite often, this is the only way to get information about program execution on the real platform.

For those who want to use their own virtual machine image, you can try the WSIM binaries for Linux[7].

Trace analysis

Although *gtkwave*[8] is not specifically for the MSP430, it is a very useful tool to analyse the traces created by the WSIM simulator. The trace file of the WSIM simulator has to be converted with the program *wtracer* to the VCD format, which can be processed by *gtkwave*. In case *wtracer* terminates with a segmentation fault, the trace file was probably not closed correctly by the simulator. With the time mode option of the simulator, you can prevent that the simulator is terminated incorrectly.
Within *gtkwave*, the following commands are quite useful:

- right-drag in signal panel
  Zoom in on the selected part of the signal

[7] `http://www.win.tue.nl/san/education/2IN26/wsim-binaries.tar.gz`
[8] `http://gtkwave.sourceforge.net/`

- left-drag in signal panel
  Measure the time between two points in the signal.
- alt-F
  Zoom out to see the complete signal.
- left-panel (SST and Signals)
  Select the signal to be shown in the Waves panel and in the Signals panel directly to the right of the Waves panel.
- The blue arrows
  Move the primary marker (the vertical red line) to the next/previous edge of the signal selected in the Signals panel.
- From the *File* menu: *Reload Waveform.*
  If you changed something in your program you can use this command to refresh the waveforms without having to select the different signals again. Obviously, the *.vcd* file needs to be regenerated first.
- From the *Markers* menu: *Drop Named Marker*, *Show-Change Marker Data* and *Collect All Named Markers.*
  The first command inserts a marker at the current position of the primary marker. The second command displays a window with the values of the markers (so this can be used to copy data into your report or into a calculator). This window needs to be closed in order to do anything else in *gtkwave.* The third command deletes all the named markers you dropped.

When LEDs are used to measure timing behaviour, you have to take into account that the LEDs can toggle on each instruction. As a result, multiple toggles might appear as a single toggle on the signal panel, so you have to zoom in sufficiently in case of doubt.

When you traced the program counter, the signal will be difficult to interpret. As a first aid, you can modify the data format of the signal by right-clicking on the name of the signal, and selecting the data format 'Analog - Step'. With 'Insert Analog Height Extension', the signal becomes easier to interpret.

It is also possible to attach a 'translate filter file' to the data format. You can use the maketff.sh[9] script to convert an ELF file into a file that *gtkwave* can read. By attaching this translate filter file to the program counter, *gtkwave* can show the name of the active function. After enabling 'View - Show Mouseover', a tooltip will appear with the translated value whenever you press a mouse button and move over the signal.

The way to attach the filter file to the PC is as follows:

- Right click *PC[15:0]* in the *Signals* part of the UIF or use the Edit menu and select *Data Format*, *Translate Filter File*, *Enable and Select*

[9]`http://www.win.tue.nl/san/education/2IN26/maketff.sh`

- In the *Filter Select* window, click *Add Filter to List*
- Select *SchedTest.tff* in the *Location* text box of the *Select Signal Filter* window and click OK
- Right click *PC[15:0]* in the *Signals* part of the UIF and go through the menu to *Enable and Select* again, as above.
- Select *SchedTest.tff* in the *Filter Select* field and click OK.

You will see the function names appearing in the *Waves* part of the UIF.

You can save the layout of your *gtkwave* signal/window configuration with the 'File - Write Save File' option, and specify the created file as additional argument on the command line. That way, you can more quickly inspect the resulting traces from a simulation.

Installing additional software

To install additional software (like another text editor), you can use the apt-get command, like:

```
sudo apt-get install application_name
```

Since the Linux environment is based on a older, minimal Ubuntu release, the applications might not be the latest versions and certain applications might require a large collection of support libraries.