lea.schoenberger [☺] tu-dortmund.de
benjamin.glaeser [☺] tu-dortmund.de
niklas.ueter [☺] tu-dortmund.de
mikail.yayla [☺] tu-dortmund.de

Exercise for the lecture
Embedded Systems
Wintersemester 17/18

Exercise Sheet 8

(10 Points)

Lab exercises for the period from Wednesday, 13th December 2017

For this exercise sheet, we use the virtual machine **RTEMS-de**.

Background

RTEMS (short for "Real-Time Executive for Multiprocessor Systems") is an open-source operating system for embedded real-time systems. In the 1980s, it has been developed for military purposes, but nowadays, it is widely used throughout multiple application areas, e.g., by the rover developed for the ExoMars mission that should start in 2020 as well as by the NASA's Solar Dynamics Observatory.

Pertaining to this exercise sheet, we unfortunately deal with terrestrial applications: RTEMS implements many aspects of real-time scheduling which shall be demonstrated in practice on a Raspberry Pi. Unfortunately, RTEMS is quite complex, but nevertheless, a detailed documentation can be found online[1].

Preparation

Hardware

We use a Raspberry Pi with an add-on Pibrella board (Figure 2), where the Raspberry Pi is controlled via serial interface a USB-serial-converter (Figure 1).

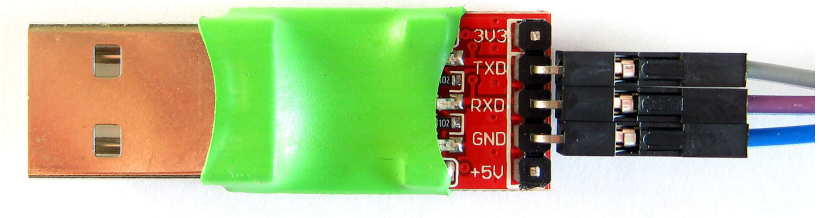


Figure 1: USB-serial-converter (color of the heat-shrink tubing may differ)

The Raspberry Pi and the USB-serial-adapter are connected via the three-pin header soldered to the Raspberry Pi, which is indicated with "GND/TXD/RXD" in Figure 2. For historical reasons, there exists two versions: One with the pin header on the edge of the board and one with the pin header pointing to the back of the USB ports. The pin header is connected to the serial adapter's GND/RXD/TXD pins via the short colorful cable (colors may differ), whereat one device's TXD port (transmit) must be connected to the other device's RXD port (receive). Please make sure to not break off the pin header since solder joints are barely resilient. Furthermore, please make sure that you use the provided USB extension cable.

The Raspberry Pi's power supply is provided via a USB cable that can be connected either to the Pibrella board's USB port (visible in Figure 2) or to the Raspberry Pi's USB port (not visible in Figure 2). As soon as the power supply is established, a red LED should light on the Raspberry Pi as well as a blue or pink LED on the Pibrella board.

[1] e.g. `https://docs.rtems.org/doxygen/cpukit/html/modules.html` and `https://docs.rtems.org/doc-current/share/rtems/html/`

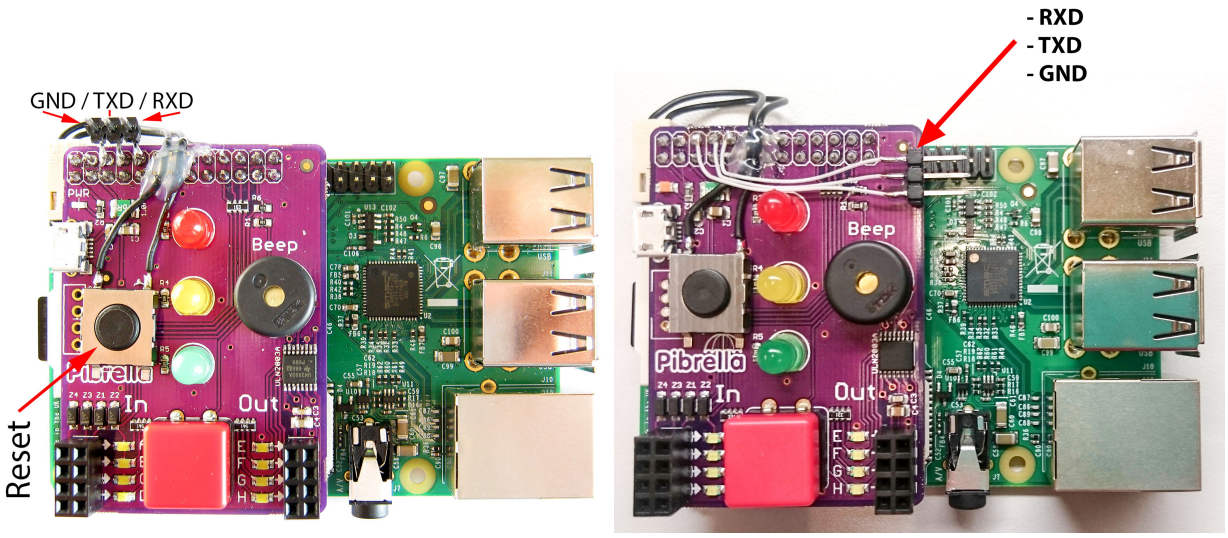


Figure 2: Raspberry Pi with two Pibrella variations (shape of the reset button may differ)

Please make sure the micro SD card is inserted properly (it should protrude no more than 2mm from the edge of the board).

Software

Due to technical reasons, RTEMS must be compiled by each user individually. We assume that you are able to use a Unix/Linus command line. As an *initial* preparation of RTEMS, execute the following commands in the given sequence:

1. `rtems-setup`
2. `cd $HOME/rtems/build`
3. `make -j4 install`

The first command extracts the archive containing the RTEMS source code to the subdirectory `rtems` in your home directory. In this directory, three more subdirectories can be found: `rtems-gpio` contains the RTEMS sourcecode, the compiling takes place in `build`, and the compiled RTEMS is installed in `install`. The source code relevant for this exercise sheed is available in the subdirectory `rtems-gpio/testsuites/samples/blatt8`, while `build/arm-rtems4.11/c/raspberrypi/testsuites/samples/blatt8` contains the compiled version of the source code.

Execute the program `blatt8` in the given *build* subdirectory to check if hardware and software work properly. In the course of this, we use the tool `raspbootcom`, which is a serial terminal combined with a code upload tool compatible with the bootloader provided on the Raspberry Pi's SD card, i.e., it transmits a keyboard input from the PC to the Raspberry Pi via the serial interface and shows the Raspberry Pi's output in the console window. When pressing the Raspberry Pi's reset button, `raspbootcom` automatically loads the program provided at that respective point in time onto the Raspberry Pi, where it is started subsequently. To execute `blatt8`, enter `raspbootcom /dev/ttyUSB0 blatt8.ralf` in the command line (make sure you are in the same directory in which the file `blatt8.ralf` is located). This example programm should output the text "Hello, World!" via the serial interface every 5 seconds.

8.1 Two Tasks (2 Points)

Open provided source code (`rtems/rtems-gpio/testsuites/samples/blatt8/init.c`) with a text editor of your choice. When reading the source code, you should realize that, in contrast to a normal C-programm, no `main` function is available. This is declared in RTEMS, such that the application is started by the function `Init` which is started as

a task by RTEMS. In the provided source code, the function contains a number of initializations which may start up to four more tasks and then close the init task. However, all of these (except one) are commented out for now.

Moreover, you should realize that another (unused) task `Task1` is defined at this point, that periodically occupies the CPU for one second, which is indicated by the flashing red LED (function `cpu_busy_loop`).

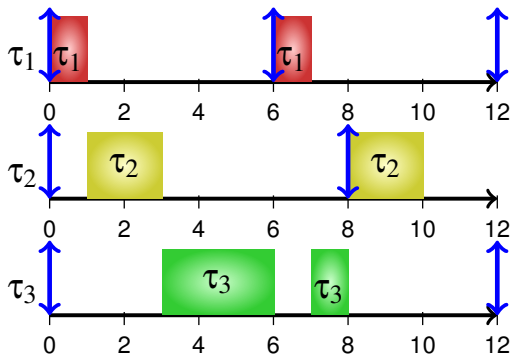
Assignment: Make sure that `Task1` is executed in addition to the already active `PrintTask`.

Hint: To recompile the modified soruce code, it is sufficient to execute the command `make` (no further arguments!) in the *build* directory.

8.2 Rate-Monotonic Scheduling (5 Points)

To demonstrate rate-monotonic scheduling, the application should be extended such that it executes three tasks according to the following schedule:

$\tau_1 = (1,6,6)$, $\tau_2 = (2,8,8)$, $\tau_3 = (4,12,12)$. [(C_i, T_i, D_i), all values in seconds]

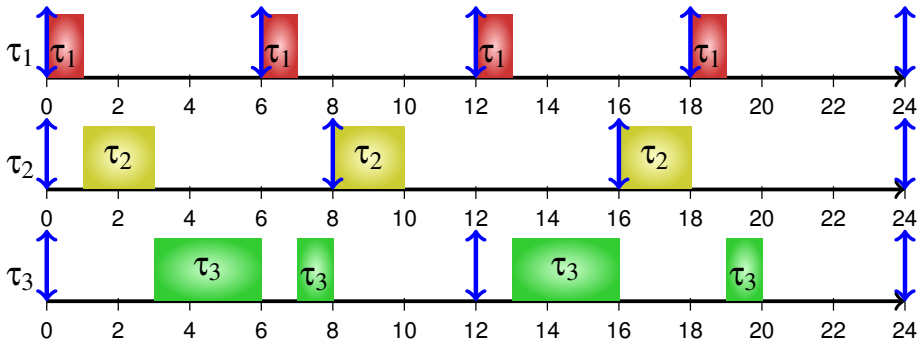


Hinweise:

- Copy & Paste may be useful.
- `LED_RED` can be replaced by `LED_YELLOW` or `LED_GREEN`.
- The number in front of the “xxxx: ... rechnet noch y Sekunden” notifications is the number of system ticks passed until the notification is displayed. Since all tasks are started at tick 0 and the system uses 100 ticks per second, the time can be easily deduced from these notifications.

8.3 Detail Problems (3 Points)

Compare the output of your program resulting from the previous assignment with the given schedule – probably a minor deviation is obvious at one point. This becomes more clear when considering not only the first 12 but the first 24 seconds:



If your program exhibits a deviation from the theoretical schedule: Can you identify the cause? It is not necessary that you correct your program.

In case you cannot detect such a deviation: Can you list possible implementation errors that can lead to such a behavior? It is not necessary that you add such an error to your program.

Hint: It may be useful to sketch the observed schedule of your program.