TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur
Betriebssysteme

Embedded Software Lab

Lab 2 - RASPNet layer 1 und 2

Christine Jakobs, Martin Richter

In this lab, you will implement the first two layers of the RASPNet protocol. They must be created as software for the ATMega microcontroller on the Gertboard. It is recommended to implement sending and receiving code parts at the same time, in order to allow testing from the very beginning. At the end of this lab, you should have a complete implementation of layer 1+2 that allows your node to send messages to itself.

Initial Remarks

The tasks should be solved in the given order. If you do not manage to solve all tasks in preparation of the next lab meeting, make sure that you solve them afterwards. Your final grade at the end of the semester depends on the availability of *all* solutions in your code repository.

The presentation of your results during the lab slot is oral. There is *no need for preparing slides* or any other kind of written material.

Be ready to answer questions. Have your code, positive and negatives experiences and according documentation available.

For multi-person teams, all must participate in the presentation and discussion of results.

For the coding part, please avoid playing with the boot / read / write lock bits, playing with the Fuse bits or writing to the boot loader memory. Writing to EEPROM regions is also not needed at the moment.

From you literature research in lab 1, you should have gathered the following facts about the Gertboard pins:

- The PC*, PD* and PB* pins on the left side expose the pins of the ATMega microcontroller.
- GND is the pin for grounding the overall setup.
- The B* jumpers make sure that the ATMega PB* pins can produce output signals on the LEDs.

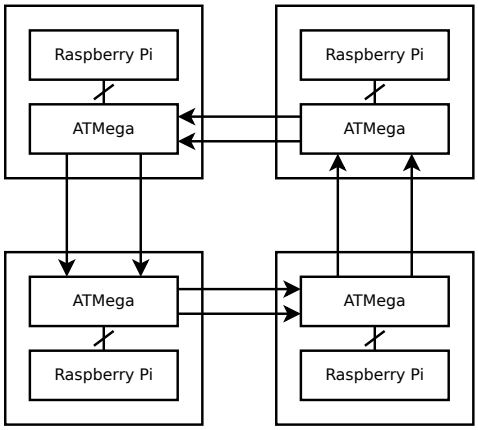

Figure 1: RASPNet network topology.

- GP8 - GP11 are the mapped GPIO pins from the RaspberryPi. The are wired with the ATMega SPI pins, so that programming becomes possible.
- GP14 and GP15 are connected to MCTX and MCRX, so that the RaspberryPI has a serial device that maps to the ATMega chip.

Task 2.1

RASPNet is designed for a logical ring architecture, where each communicating node is connected to exactly two other nodes in the network. Together, all nodes form a ring structure as shown in Figure 1.

Compare the idea of ring topologies for a network with tree topologies (such as in switched networks) and bus topologies (such as with CAN). What are the advantages and disadvantages with respect to:

- Average round-trip time
- Bandwidth
- Node failures
- Interconnect failures
- Predictable timing

Task 2.2

Prepare the Gertboard wiring according to Figure 2. If a wiring already exists, check it for correctness.

This wiring scheme connects all relevant communication lines to the node itself, so that a communication ring with one node is created. The pins are used in the following way:

- PB1: Clock send

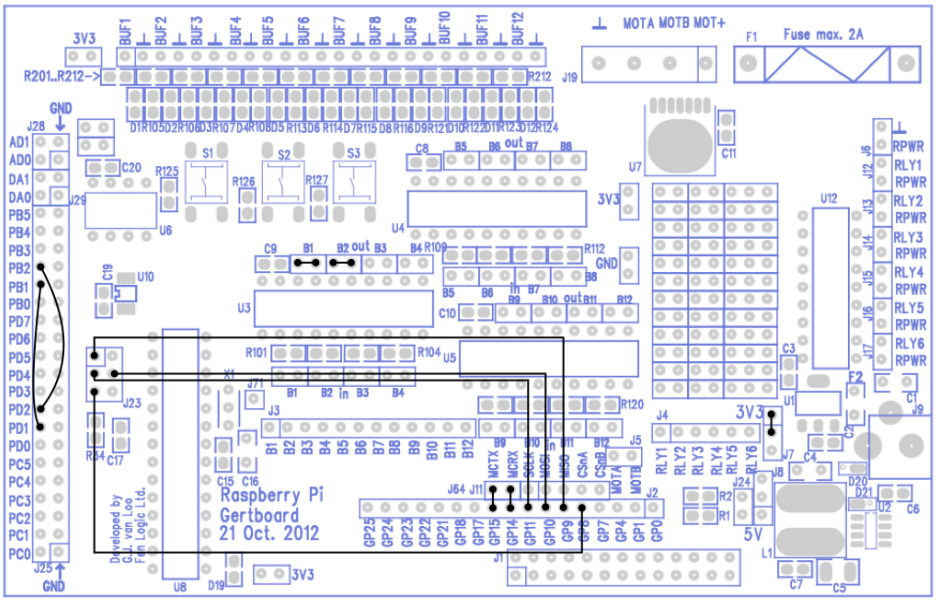

Figure 2: Gertboard wiring scheme for layer 2 development.

- PB2: Data send
- PD1: Clock receive
- PD2: Data receive

Task 2.3

Write a program for the ATMel that uses timer interrupts to output a character pattern on the serial connection (see Task 1.6). You timer interrupt handler should output the character 'I', while the main program prints the character 'M' in an endless loop.

Note:

The ATMega processor has three hardware timers. Each of them allows to periodically generate an interrupt after a given amount of time. An interrupt stops the currently running instruction stream and execute handler code that is called *Interrupt Service Routine (ISR)*. When the ISR is finished, the processor resumes the normal code execution.

We recommend to start with a frequency of 1 interrupt per second, which is extremely slow. This allows you to have a visual control of what is going on in your program.

In the ATMega processor, the list of ISRs is managed as table starting at 0x00 in main memory. Each ISR table entry conatins a jump instruction to one specific ISR. If an interrupt occurs, the processor looks at a specific table index in memory and executed the code there, therefore jumping directly to the ISR.

AVR-GCC supports the development of ISRs with the `ISR(...)` macro. This macro allows to specify an ISR implementation for a particular interrupt type:

```
#include <avr/interrupt.h>

ISR(ADC_vect)
{
    // user code here
}
```

Consult the ATMega documentation [2] and other sources [1, 4] for further details about interrupt types and the configuration of timer interrupts. What does the prescale-unit do and how does it function? How can you use the timer interrupt feature to run code periodically with a particular frequency?

Task 2.4

Implement the clock signal for RASPNet layer 1, as described in the protocol specification [3]. The clock signal should be send on `PB1` and triggered according to a predefined clock frequency. Indicate the receiving of a clock signal on `PD1` by using the Gertboard LEDs.

Note:

Use the ATMel documentation to learn about the possibilities of a pin-change interrupt handler.

Task 2.5

Try to modify the clock frequency of the communication. What happens?

What happens in your implementation when there is no data to send?

Note:

In preparation for later tasks, it is reasonable to introduce separate send and receive buffers. For this task, they only need to hold one bit, but this will change later.

You must consider that both buffers are read and written by multiple activities. Prepare your code so that only one activity (interrupt handler / main code) can modify the buffer at a time.

It will be helpful to have a static send buffer with pre-filled data for testing.

Task 2.6

Implement the layer 2 sending part, in accordance to the RASPNet specification [3]. You code should fill a send buffer with valid frame data, therefore, your send buffer implementation must now handle all bits for a frame. When the send buffer is ready, the bit sending code from your layer 1 implementation should be triggered.

Note:

The details of the CRC implementation are tricky. We recommend that you develop the neccessary C code for generating checksums in a desktop application project and move it to your RASPNet project afterwards.

Task 2.7

Implement the layer 2 receiver part, in accordance to the RASPNet specification [3]. Test your implementation with a *predefined frame* that is just copied to the send buffer.

The receiver must compute the CRC checksum of the received payload and compare it to the checksum being sent with the frame data. Print the result of your receiver activity as status message on the serial console.

Note:

One implementation strategy is to exend the pin-change ISR to recognize frame starts and (with the use of the size-field in the package) frame endings. You need to extend your receive buffer implementation accordingly.

If you run into problems with receive buffer overflows, try to reduce the clock frequency.

References

[1] Timers on the atmega168/328. `https://sites.google.com/site/qeewiki/books/avr-guide/timers-on-the-atmega328`.

[2] Atmel. Interrupts - AVR Libc Reference Manual Modules. `http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__interrupts.html`.

[3] Stefan Naumann. RaspNet - A simple realtime network protocol for microcontrollers. `https://osg.informatik.tu-chemnitz.de/lehre/emblab/raspnet.pdf`.

[4] University of Washington. Atmega328 timers and interrupts. `http://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture7.pdf`.