Lab 3

Radio-controlled Clock with DCF77 Interface

Bearbeitet durch:

Datum: **Hinweis: Je Gruppe ist eine Ausarbeitung gefordert**

Mit Ihrer Unterschrift bestätigen Sie, dass Sie die Vorbereitung zum Laborversuch selbstständig durchgeführt und den Bericht in Ihrer Laborgruppe ohne fremde Hilfe angefertigt haben. Falls Sie dennoch Lösungen aus fremden Quellen (Bücher usw.) verwenden oder bei der Vorbereitung mit anderen Laborgruppen zusammengearbeitet haben, müssen Sie diese Quellen im Bericht eindeutig angeben (welche Lösungen (Programme/Doku) wurden kopiert, woher wurden Sie kopiert (Name der Autoren, Link) usw.). Bei mehr als 10% fremden Quellen, bei fehlenden oder unvollständigen Quellenangaben oder ohne Unterschrift auf dieser Seite kann der Laborversuch nicht anerkannt werden. In diesem Fall muss das gesamte Labor wie bei jeder nicht bestandenen Prüfung im folgenden Semester wiederholt werden.

Unterschrift aller Gruppenmitglieder:

Contents

Additional information:

- Lecture manuscript chapters 1 - 4 and Appendix CodeWarrior HCS12 Development Tools
- Documentation package for the HCS12 Microcontroller and the Dragon12-Board

1. Overview

Purpose of this Lab

- Implementing a complex application
- Mixed programming in C and assembler

In the second lab we developed a clock. The clock was driven by a timer, which triggered an interrupt every 10ms. A software counter in the ISR creates a 1s tick, from which minutes and hours were derived. Unfortunately, this clock loses its time value, when turned off, and thus must be set manually when turned on. Additionally, because the quartz oscillator's precision is limited, the time will drift slightly, when the clock programs runs a long time. Another

drawback is that the clock must be manually set when time changes from standard time (MEZ) to daylight savings time (MESZ) and vice versa.

In this lab 3, setting the clock shall be done automatically by using the DCF77 radio signal, which transmits time and date information.

Structure of this Lab

The lab consists of a preparation part and a lab part. The prep part includes a number of **Prep Tasks**, which must be solved before you come to the lab. In most cases these are subroutines, which must be designed, developed and tested with the simulator/debugger. In the lab you will have to demonstrate, that these subroutines work on the Dragon12 board, before you use them within the **Lab Task** programs, you have to develop in the lab.

(1) **Required knowledge and time for preparation**: Plan enough time to prepare for the lab and carefully test the subroutines written. When you come to the lab, you must be able to write larger programs and debug them with the CodeWarrior toolchain. There will not be enough time in the lab, if you start from scratch or need to read the documentation or search the online help. If you are not accordingly prepared, you will get enough time for preparation, because you will then have to retry the lab in the next semester.

(2) For all preparation programs you have to bring the program design documents (e.g. code structure, flow charts) and the CodeWarrior projects to the lab. All students in your group must be able to demonstrate and explain your programs in detail. Your programs must be fully commented. Without documentation and comments or if you fail to explain your code, you will not pass the lab!

(3) If you use the help of others (e.g. copy code from other sources than the lecture and lab code templates) you must clearly state this in the documentation and in the program comments. In this case, you must exactly specify, which parts of your code have been copied (even if you modified this code). If the code of more than 10% of your program has been designed with the help of others or if you did not clearly state, that you copied code or design documents, the lab instructor may declare the lab as failed. With your signature on the name list in the lab you explicitly confirm that you did work according to these rules.

Sicherheitshinweise

- Zuständige Personen im Labor Informationstechnik
  - Laborleiter: Prof. Dr. Zimmermann Tel. 4227
  - Laboringenieur: Robert Zins Tel. 4263
- Aufgaben der Studierenden
  - Versuche nach Versuchsanweisung und Vorgaben der Labormitarbeiter durchführen
  - Bei Störungen an Geräten und Installationen Laboringenieur, Labormeister oder Laborleiter verständigen.
  - Keine Eingriffe in Geräte vornehmen, vor allem nicht an spannungsführenden Teilen.
  - Laborvorbereitung und Labordurchführung nur in Anwesenheit von mindestens zwei Personen im Labor.
- Verhalten bei Verletzungen, Unfällen und Feuer
  - Bei Unfällen mit elektrischen Geräten oder Feuer elektrische Geräte sofort abschalten, Notaus betätigen.
  - Bei Kleinverletzungen Ersthelfer aufsuchen (Aushang bzw. Sicherheitsordner im Labor), Krankentrage im Flur, Sanitätsraum im Erdgeschoss
  - **Bei größeren Verletzungen Notarzt verständigen: Tel. 112**
  - **Bei Feuer Feuerwehr verständigen: Tel. 112**
  - **Hochschulinterner Notruf: Tel. 9**
  - Bei Feuer Fenster und Feuerschutztüren schließen, Gebäude über grün ausgeschilderte Fluchtwege verlassen, Aufzüge nicht benutzen
  - Bei Kleinbränden Feuer mit Pulverfeuerlöscher bekämpfen (im Labor und auf den Gängen), falls keine Vergiftungs- oder Erstickungsgefahr besteht.
- Allgemeines
  - Arbeitsplatz sauber halten, Abfälle in vorgesehene Behälter entsorgen oder mitnehmen, Rauch- und Alkoholverbot beachten
  - Beim Verlassen des Labors PCs und Geräte abschalten, Fenster und Türen schließen, Stühle aufräumen

Weitere Sicherheitsinformationen finden Sie im aushängenden Sicherheitsordner im Labor.

2. Format of the DCF77 Signal

DCF77 is a low frequency radio transmitter, operated by Physikalisch-Technischen Bundesanstalt PTB. The transmitter is located near Frankfurt. Its signal can be received in most parts of Europe. The signal uses a digital amplitude modulation, which can be received by a simple antenna and decoded with a low pass filter and a comparator. In the lab we use an antenna mounted at the windows of the room and a decoder, which outputs a pulse signal, which contains the time and date information in serial BCD code. Decoding this information must be done by your software.

When idle, the digital DCF77 signal (Fig. 2.1) is H (High). Every second the pulse goes to L (Low) (second impulses). Each L impulse contains 1bit of the time and date information. To mark the begin of a new minute, the last second impulse (bit 59) before the begin of a minute (bit 0) is missing (minute mark).

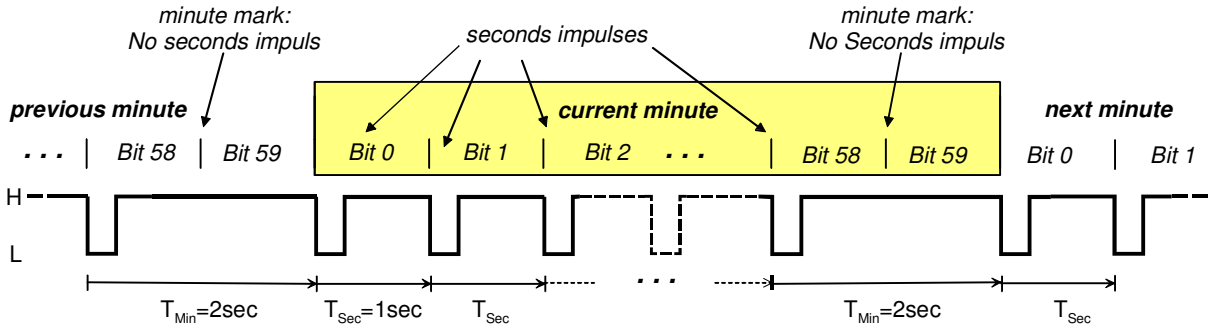

Fig. 2.1 DCF77 format: Second impulses and minute marks

The bits of time and date information are coded via the length of the L-phase of the second impulses (Fig. 2.2). A short L-impulse of approximately 100ms codes a '0' bit, a longer L-impulse of approx. 200ms stands for a '1' bit.

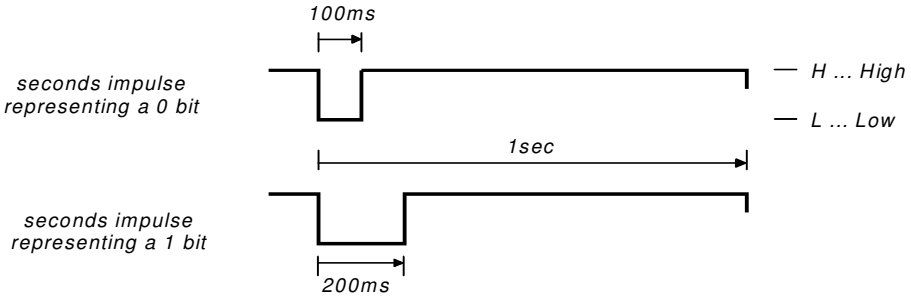

Fig. 2.2 Coding of 0 and 1 in second impulses

The 59 data bits transmitted each minute contain the current date and time plus additional control information (for details see [1] and Fig. 2.3):

Second / Bit	*Information*
0 to 16	Not used in the lab
17, 18	01_B = normal time MEZ, 10_B = daylight savings time MESZ
19	Not used in the lab
20	Always 1_B
21 to 27	Minutes (7bit BCD)
28	Even parity for bits 21 to 27

29 to 34	Hours (6bit BCD)
35	Even parity for bits 29 to 34
36 to 41	Day (6bit BCD)
42 to 44	Weekday (3bit) with 1=Monday, …, 7=Sunday
45 to 49	Month (5bit BCD)
50 to 57	Year (8bit BCD, only the last two digits, i.e. without century)
58	Even parity for bits 36 to 57

All values are BCD coded and transmitted starting with the LSB. The number of bits is chosen as small as possible. E.g. days have a range of 1 to 31, which requires 6bit in BCD (2bit for the 10's digit, 4bit for the 1's digit). The year's range is limited to 0 to 99 requiring 8bit (4bit each for the 10's digit and the 1's digit), the century is not transmitted.

Example:
The 25th day of a month is coded as follows:

$$25_D = 20 + 5 = \underset{\text{MSB}}{1}\ 0\ .\ 0\ 1\ 0\ \underset{\text{LSB}}{1}\ _{\text{BCD}}$$

The transmitted bit sequence is:

Bit	36 LSB	37	38	39	40	41 MSB
Bit value	$1=2^0$	$2=2^1$	$4=2^2$	$8=2^3$	$10=2^0\cdot10$	$20=2^1\cdot10$
Coding	1	0	1	0	0	1
Impulse	200ms	100ms	200ms	100ms	100ms	200ms

Example:
Year 2007 is transmitted as:

Bit	50 LSB	51	52	53	54	55	56	57 MSB
Bit value	$1=2^0$	$2=2^1$	$4=2^2$	$8=2^3$	$10=2^0\cdot10$	$20=2^1\cdot10$	$40=2^2\cdot10$	$80=2^3\cdot10$
Coding	1	1	1	0	0	0	0	0
Impulse	200ms	200ms	200ms	100ms	100ms	100ms	100ms	100ms

Bits 17, 18, 20 and parity bits 28, 25 and 58 may be used to check the received data.

As the transmission of the complete bit sequence takes a full minute, the time for the next minute will be transmitted. I.e., in the minute beginning at 23:59 the time 00:00 and the date of the day beginning at noon will be transmitted.

Further information about DCF77 can be found here:

[1] Physikalisch-Technische Bundesanstalt:
http://www.ptb.de/de/org/4/44/442/dcf77_1.htm

[2] Langwellensender DCF77, Wikipedia, http://de.wikipedia.org/wiki/DCF77

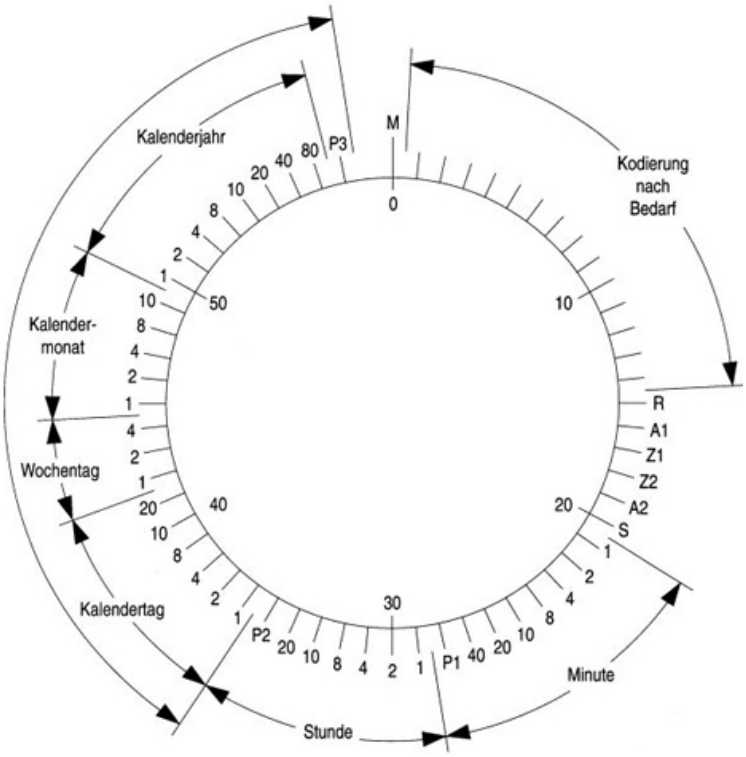

Fig. 2.3

3. Requirements for the Radio-controlled Clock

The requirements for the radio-controlled clock are:

- The first line of Dragon12's LCD display shall show the time in the following format
  `hours : minutes : seconds`
  The second line shall display the date **`day . month . year`**
  The manual *Set Mode* known from Lab 2 is not required.
- Because the DCF77 transmission fails sometimes (bad radio signal in massive steel-concrete buildings, transmitter maintenance etc.), the basic concept of Lab 2's clock is still used, i.e. the clock is driven by the timer clock. The DCF77 info is only used to set the clock once per minute and to display the date. If the DCF77 radio information fails, the clock shall continue to run based on the timer clock.
- The timer clock shall be used to toggle the LED on port B.0 once per second.
- The LED on port B.1 shall be turned on, when the DCF77 signal is Low and turned off, when the DCF77 signal is High. Thus, when the DCF77 signal is received, this LED will also be toggled once per second.
- The LED on port B.3 shall be turned on, when a complete and correct DCF77 date and time information has been decoded and turned off at once, when no or wrong data has been received. Check, if hours are in the 0 … 23, minutes and seconds in the 0 … 59, days in the 1 … 31 and months in the 1 … 12 range. Parity checks are optional. When an error is detected, your program shall turn on LED on port B.2, until valid data is received.
- The DCF77 pulse signal is connected to port H.0 on the Dragon12 board. But you should write your program in such a way, that another port can be used too. Thus, do poll the port and do not use interrupts for port H. The detection of positive and negative edges in the DCF77 signal and the timing measurement shall be done in software. Handle the initialization of the port in a function **`void initPort(void)`** and read the port in a function **`char readPort()`**, which shall return 0, if the DCF77 signal is Low and >0, if the signal is High.

4. Software Architecture for the Implementation

CodeWarrior project **`lab3-Funkuhr-Vorlage.mcp`** includes a C-implementation of a free-running clock similar to Lab 2. **`main.c`** and **`clock.c`** are written in C, but use the well-known (slightly modified) LCD and timer drivers in HCS12 assembler.

The free-running clock is distributed over three major functions (Fig. 4.1):

- Function **`tick10ms()`** is a subroutine, which is called periodically every 10ms by the ISR of the timer module. To keep the interrupt short, the function increments a software timer **`uptime`**, calls another subroutine **`sampleSignalDCF77()`**, which polls the DCF77 signal (see below) and sets global variable **`clockEvent`** once per second.
- The main loop checks this global variable and calls function **`processEventsClock()`**, when the variable has been set, i.e. once per second. **`processEventsClock()`** counts the seconds, converts them into hours, minutes and seconds and stores them in global variables **`hrs, mins, secs`**.
- Then the main loop calls function **`displayTimeClock()`**, which converts the time into an ASCII string and displays the time on the LCD display. At the end of the loop global variable **`clockEvent`** is reset.

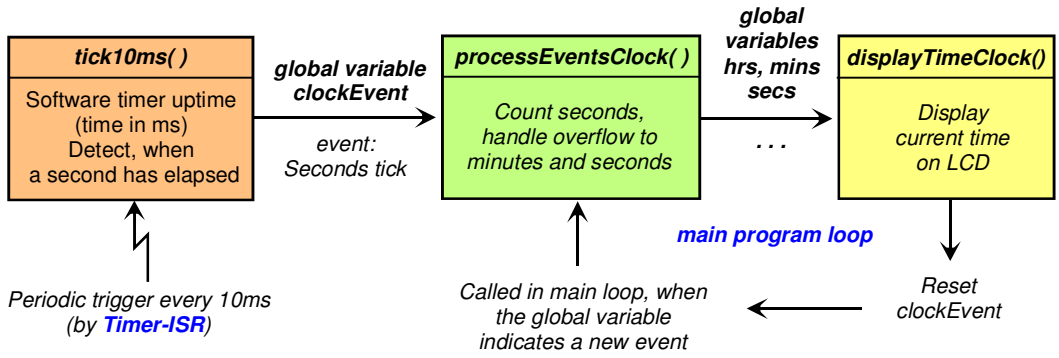

Fig. 4.1: Architecture of the free-running clock

The advantage of this structure is that the hardware dependent and time critical parts are isolated from the hardware independent and uncritical parts and that all modules can be tested independently.

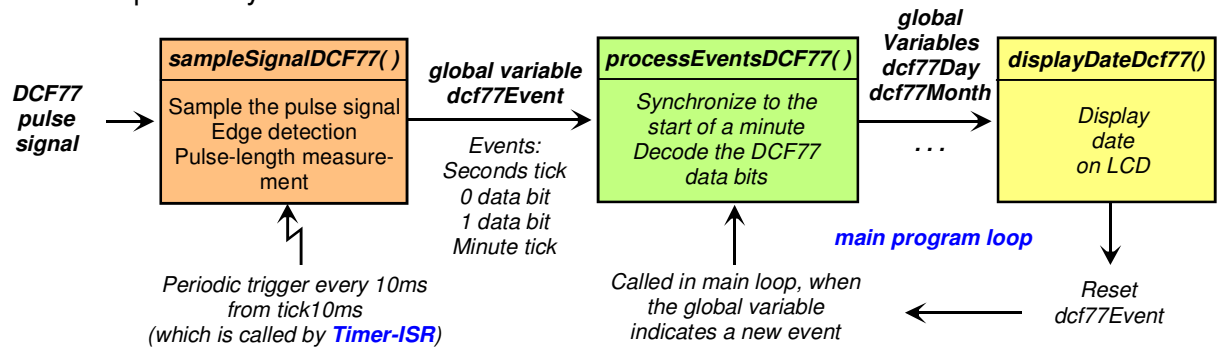

Fig. 4.2: A possible architecture for the DCF77 extension (additionally to Fig. 4.1)

The same principle is used for the DCF77 extension (Fig. 4.2):

- The DCF77 signal is sampled in a new function **`sampleSignalDCF77()`** periodically every 10ms, i.e. **`sampleSignalDCF77()`** will be called in the timer ISR. The edge detection is done by comparing the current value of the signal with the value of the last sample. Times T_{Low} und T_{Pulse} (Fig 4.3) are measure via software timer **`uptime`** (see above), which is passed as parameter, when the function is called. Rather than using the timer module's TCNT register, which has a relatively short period, the software timer has a much larger period of 65536ms. **`sampleSignalDCF77()`** detects the positive and negative edges and measured times and converts them into "events" stored in global variable **`dcf77Event`**:

Event	*Signal edge*	*Purpose and condition*
NODCF77EVENT	No	Signal did not change since last polling
VALIDSECOND	Negative	Valid second impulse, if T_{Pulse}=1s ± 100ms. The tolerance window is required, as the DCF77 signals period and/or the timer interrupt period may jitter.
VALIDMINUTE	Negative	Valid minute mark, if T_{Pulse}=2s ± 100ms.
VALIDZERO	Positive	Valid 0 bit, if T_{Low}=100ms ± 30ms
VALIDONE	Positive	Valid 1 bit, if T_{Low}=200ms ± 30ms
INVALID	Negative Positive	If T_{Pulse} is outside of the tolerance window. If T_{Low} is outside of the tolerance window.

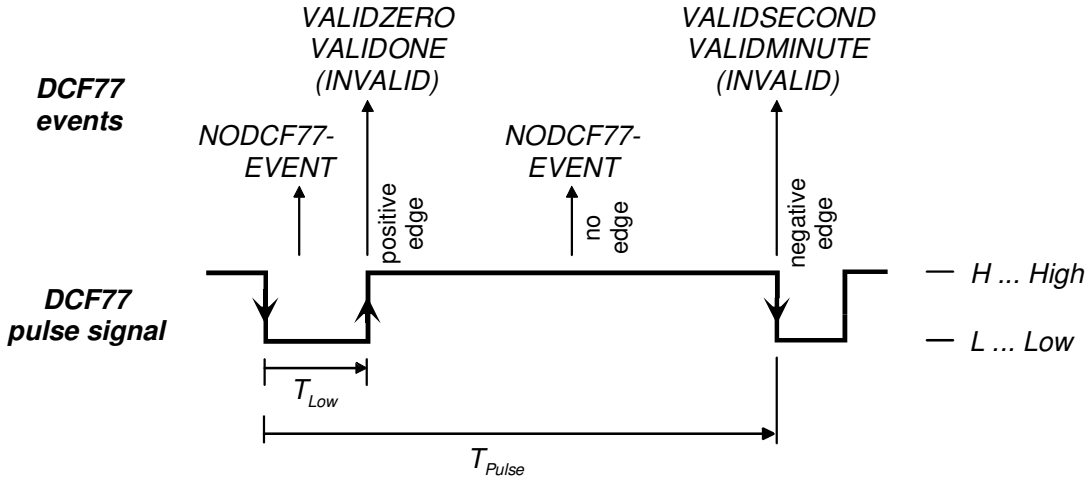

Fig. 4.3: DCF77 events

- Function **`processDCF77Events()`** is called in the main loop, whenever an event (other than NODCF77EVENT) has occurred. The function stores the associated DCF77 data bit. When the information is complete, i.e. when bit 58 has been received, the function decodes the time and date and sets the time of the free-running clock by calling function **`setClock()`**. It is recommended to design function **`processDCF77Events()`** as a Finite State Machine. Use the DCF77 bit number as the state and react on the events.
- Finally, in the main loop function **`displayDateDCF77()`** is called to display date and time, and global variable **`dcf77Event`** is reset.

Prep Task 4.1: Program Design and Implementation

Design your software architecture and describe it graphically similar to Fig. 4.1/4.2 plus program flow charts, data dictionary etc. (see documentation requirements in Lab 2).

The architecture presented here shall be considered as a proposal only. You may come up with a completely different solution, as long as it fulfills the requirements.

We expect you to **develop AND document** (see documentation requirements in Lab 2) the radio-controlled clock on your own and **come to the lab with a fully tested solution**. You may use and modify our template **`lab3-Funkuhr-Vorlage.mcp`** plus your code from Lab 1 and Lab 2.

You may write your program in C and/or HCS12 assembler.

Try to test as much as possible with the simulator. It is very difficult to test the program on the Dragon12 board directly, as the time constantly changes and the DCF77 signal does not stop, if you single step your code or halt on a breakpoint:

- To help with debugging, our project template in module **`dcf77Sim.c`** includes a function **`readPortSim()`**, which simulates the DCF77 signal (without hardware). Use this function in the simulator (rather than function **`readPort()`**, which will read the real DCF77 signal on port H.0). **`readPortSim()`** must be called periodically every 10ms and returns a logic ‚0' or ‚1', which represents the (simulated) DCF77 signal.
- Please remember, that the simulation may run faster or slower than real-time. But even when you run on the Dragon12 board in real-time, decoding DCF77 may take up to two minutes from starting your program until you have valid date and time information. Thus, developing the program with a trial-and-error approach is not a good idea and will cause days, if not weeks. You need to carefully design your program and then incrementally test all functions individually, before you integrate them.
- Finally test your program on the Dragon12 board.

Prep Task 4.2: Temperature Measurement, AM/PM, Weekdays

Add the following features to the radio-controlled clock:

- Display the weekday (Sun(day), Mon(day), …) in front of the date.
- The room temperature shall be measured and displayed according to the requirements in Lab 2.
- Whenever button SW2 is pressed for 1sec or more, the time display shall be toggled between 12h and 24h mode. In 12h mode, “AM” or “PM” shall be added in the time display.
- Test your program on the Dragon12 board.

Lab Task 4.3: Project Presentation

We expect you to come to the lab with a fully working program and complete documentation. You will get approx. 20mins to show your working program, present its architecture and code implementation and hand over a printout of your documentation in a folder to the lab supervisor. Please do not forget to include this lab manual in the documentation and sign the disclaimer on the front page.

At the end of this project presentaton, you shall **upload your Codewarrior source files** for lab task 4.2 to our server. Go to web page

http://www.it.hs-esslingen.de/~zimmerma/vorlesungen/ca3/local/CAupload.htm

and follow the instructions. Your files will be stored on the server and analyzed. To upload your files, you need the user name and password you got to download the lecture manuscript and lab manuals.

Hints: Using the DCF77 Receiver in Lab F1.307

1. DCF77 radio receiver box
   - The power supply adapter for the radio receiver (grey box on top of the power outlets between places 5 and 6 in the lab) shall be plugged in the outlet marked in red.
   - If a DCF77 signal is received, the red LED on the front of the grey box must blink once per second.

2. Connection between radio receiver box and Dragon12 board
   - Connect the round black DIN-plug (arrow on top) to the black socket near the potentiometer on the Dragon12 board.
   - The digital DCF77 pulse is connected to port H.0.
   - You cannot use button SW5 in your program, because the button is connected to the same port pin.

3. Testing the DCF77 radio signal
   - Use CodeWarrior project DCF77Test and run it on the Dragon12 board.
   - This program polls port H.0 and outputs its value to LED on port B.1. If everything is correct, the LED should toggle every second.
   - If it does not work, check steps 1 and 2.

4. If the radio signal or the receiver box does not work
   - Use function **`readPortsim()`** to simulate the DCF77 signal. The function works in the simulator/debugger and on the Dragon12 board, but delivers outdated date and time values.