HOCHSCHULE ESSLINGEN

Lab 3: Radio-controlled Clock with DCF77

Goals:

In Computer Architecture you will learn the architecture, programming & debugging of embedded systems: in the end you've developed a radio-frequency controlled clock based on the DCF77 signal.

In the first lab, You learn how to program & debug software using Assembler, laying the ground-work for the later labs: learning how to control the UI based on LEDs, the LCD display and buttons as input.

In Lab 2, you'll develop the core of the clock, using the HCS12's timer module and some buttons, to let the user set the initial time. Additionally, the clock will use the analog-to-digital converter (ADC) to measure the room temperature.

In Lab 3, you'll extend the clock with a DCF77 radio-frequency interface, receiving the current date and time information via antenna, so that the clock automatically sets time, switches between normal time and daylight savings time and tracks the date and leap years correctly.

Successful completion:

In each Lab You each must deliver & present working software. Additionally in Lab 2 and Lab 3 you must deliver a lab report with documented software.

Please mark in Your source code, who worked on what parts of the code. The default is groups of **two persons**, any code copied (other than from the samples) must be marked.

Format of the DCF77 signal

DCF77 is a low frequency radio transmitter, operated by the Physikalisch-technische Bundesanstalt PTB. The transmitter is located near Frankfurt. Its signal uses digital amplitude modulation, which can be received by a simple antenna and decoded with a low pass filter and a comparator. In our lab we use an antenna mounted at the windows of the laboratory 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 (see Fig. below) is H (High). Every second the pulse goes to L (Low). Each L impulse contains one bit of the time and date information. To mark the beginning of a new minute, the last second impulse (bit 59) before the start of a minute (bit 0) is missing (minute mark).

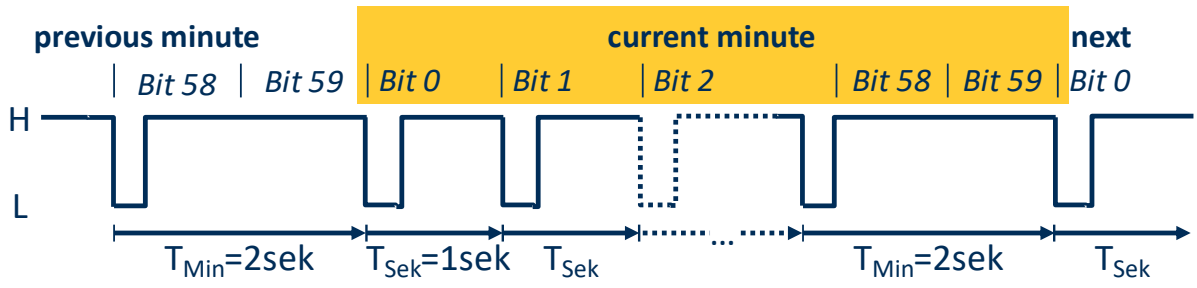

Figure 1 DCF 77 format: Second impulses and minute marks

The lack of seconds pulses marks the end of a minute pulse. Encoded in the seconds pulses the information for date and time via the length of the L-phase: a short L-impulse of approximately 100ms codes a '0' bit, while a longer L-impulse of approximately 200ms encodes a '1' bit as shown in Figure 2.

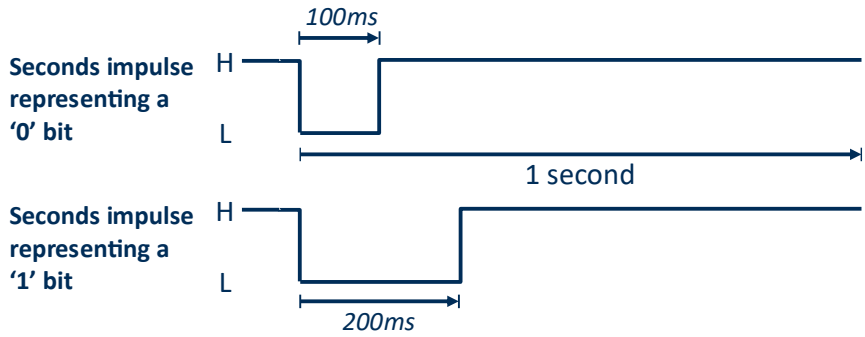

Figure 2 Coding of 0 and 1 in seconds impulses

The 59 data bits transmitted each minute contain the current date and time plus additional control information according to the following table (for details see[1] and Figure 3).

Second/Bit	Information
0 to 16	Not used in the lab (alarm system, and commercial weather forecast data)
17, 18	01B = normal time MEZ, 10B = daylight savings time MESZ
19	Not used in the lab
20	Always 1B
21 to 27	Minutes (7 bit BCD)
28	Even parity for bits 21 to 27 (P1 in Figure 3)
29 to 34	Hours (6 bit BCD)
35	Even parity for bits 29 to 34 (P2 in Figure 3)
36 to 41	Day (6 bit BCD)
42 to 44	Weekday (3 bit) with 1 = Monday, …, 7 = Sunday
45 to 49	Month (5 bit BCD)
50 to 57	Year (8 bit BCD), only the last two digits, i.e. without century
58	Even parity for bits 36 to 57 (P3 in Figure 3)

[1] PTB Information on DCF77

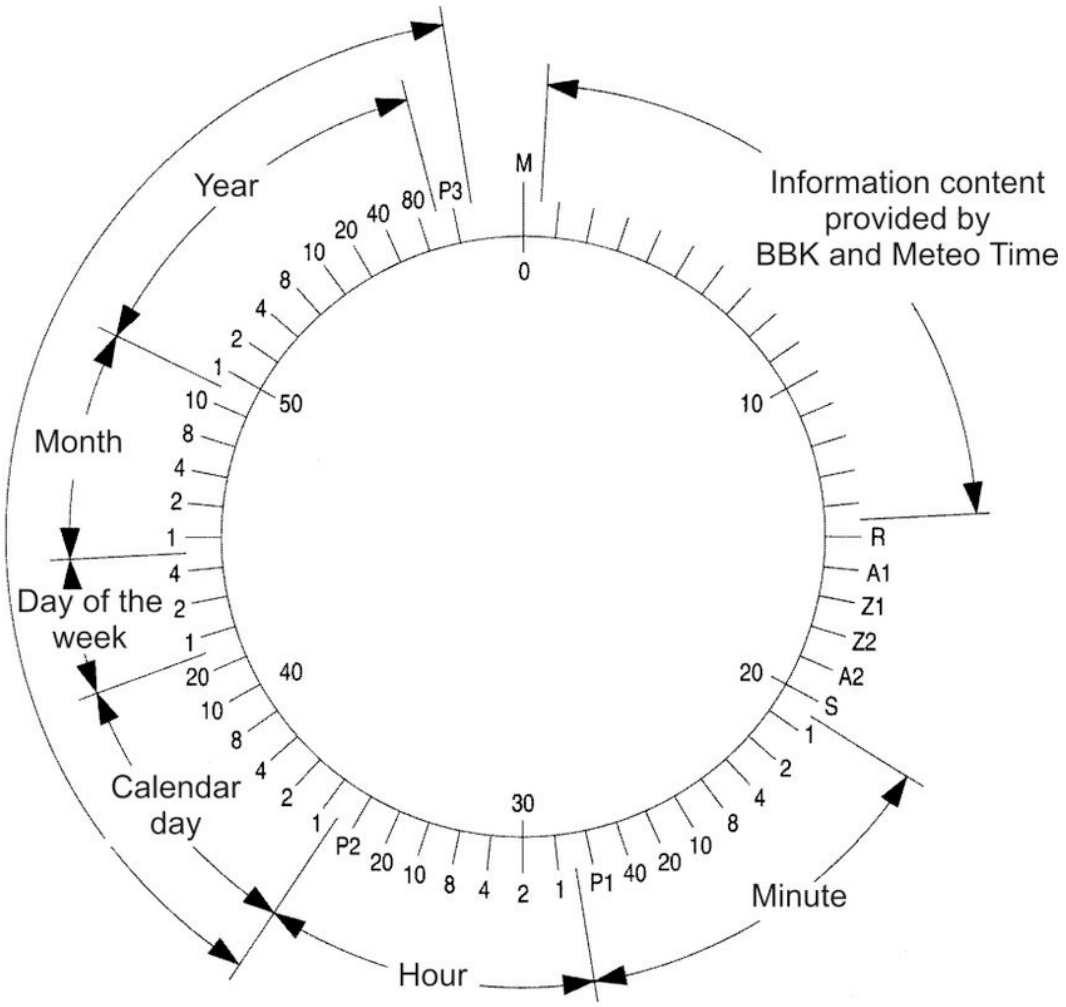

Figure 3 DCF77 encoding of minutes, etc.

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 6 bit in BCD (2 bits for the 10's digit, 4 bit for the 1's digits). The year's range is limited to the range 0 to 99 required 8 bit (4 bit each for the 10's digit and the 1's digit), the century is not transmitted.

Example:

The 25th day of the month is encoded as follows:

$25_D = 20 + 5 =$ 1 0 . 0 1 0 1
MSB LSB

The transmitted bit sequence is:

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

Example:

Year 2022 is transmitted as:

Bit	50 LSB	51	52	53	54	55	56	57 MSB
Bit value	$1=2^0$	$2=2^1$	$4=2^2$	$8=2^3$	$10=2^0*10$	$20=2^1*10$	$40=2^2*10$	$80=2^3*10$
Coding	0	1	0	0	0	1	0	0
Impulse	100ms	200ms	100ms	100ms	100ms	200ms	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. That means in the minute beginning at 23:59 the time 00:00 and the date of the day beginning at noon will be transmitted.

Requirements for the radio-controlled Clock

The requirements for the radio-controlled clock are (Note: for reference purposes requirements are numbed as [Req …]:

- [Req 1.1] The first line of Dragon12's LCD display shall show the time in the following format:
  hours : minutes : seconds
  While the second line shows the date: day . month . year
  The manual *Set Mode* known from Lab 2 is **not** required.
- [Req 1.2] Because the DCF77 transmission fails sometimes (bad radio signal in massive steel-concrete buildings, transmitter maintenance, etc.), the basic concept of Lab2'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.
- [Req 1.3] The timer clock shall be used to toggle the LED on Port B.0 once per second.
- [Req 1.4] 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 toggle once per second.
- [Req 1.5] 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, and minutes in the 0 … 59, days in the 1 … 31 and months in the 1 ... 12 range. Parity checks are required. When an error is detected, your program shall turn on LED on port B.2, until valid data is received.
- [Req 1.6] The DCF77 pulse signal on the real Dragon12 board is connected to port H.0. As the real DCF77 antenna signal may include spikes and the signal edges may jitter, the signal should be polled every 10ms rather than using port H's interrupts.

Software Architecture for the implementation

CodeWarrior project **lab3-Funkuhr-Vorlage-OS.mcp** includes a C-implementation of a free-running clock similar to Lab2. The files `main.c` and `clock.c` as well as LCD, LED and timer drivers are written in C.

The free-running clock is implemented via three major functions, see Figure 4:

- Function `tick10ms()` is a subroutine, which is called periodically every 10ms by the ISR of the timer module. To keep the ISR short, the function increments a software timer `uptime`,

calls subroutine `sampleSignalDCF77()` (see below) and sets global variable `clockEvent` once per second.

- Function `processEventsClock()` will be called by the "operating system" when the variable `clockEvent` has been set, i.e. once per second. The `processEventsClock()` counts the seconds, converts them into hours, minutes and seconds and stores them in global variables `hrs`, `mins`, `secs`. Then it sets the global Variable `displayEvent`.
- When global variable `displayEvent` is set, the "operating system" will call function `displayDateTimeClock()`, which converts the time into an ASCII string and displays the time on the LCD display.
- When functions `processEventsClock()` or `displayDateTimeClock()` return, the "operating system" will reset the respective `clockEvent` and `displayEvent`.

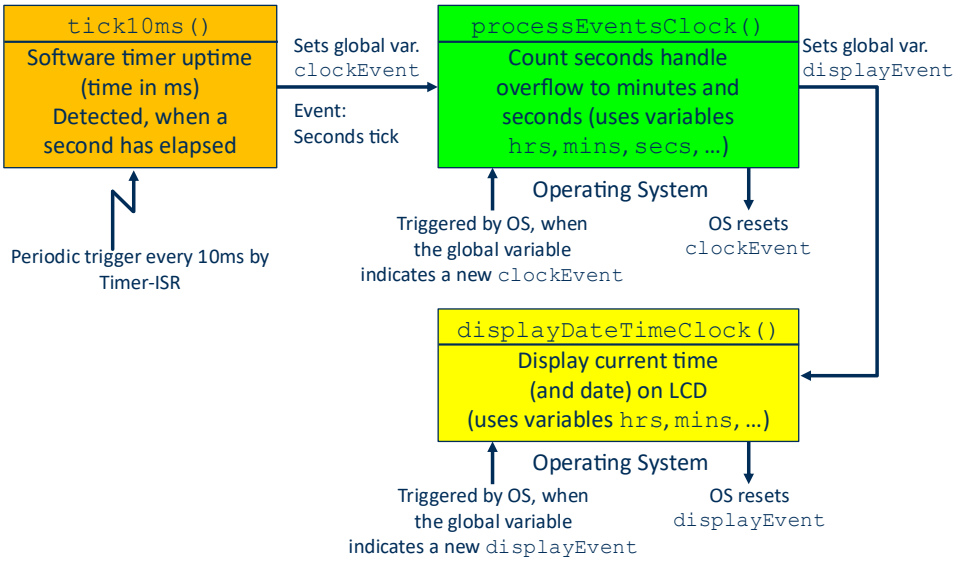

Figure 4 Architecture of the free-running clock

This way offers the advantage that hardware-dependent and time critical parts are isolated from the hardware-independent and uncritical parts and that all modules may be tested independently.

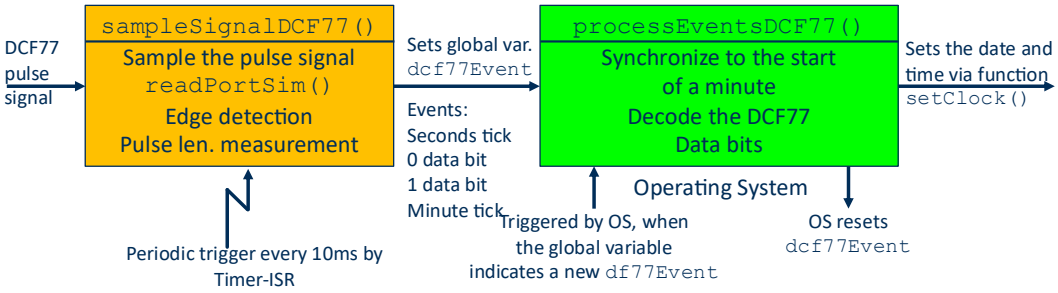

Figure 5 A possible architecture for the DCF77 extension (additionally to Figure 4)

The same principle is used for the DCF77 extension (see Figure 5):

- The DCF77 signal is sampled in function `sampleSignalDCF77()` periodically every 10ms. To debug the program without a radio antenna, the DCF77 is simulated by the function `char readPortSim()`, which returns 0, when the simulated DCF77 signal is Low and >0, when the signal is High.
- The detection of positive and negative edges in the DCF77 signal and the timing measurement is done in software by comparing the current value of the signal with the value of the last

sample. Times T_{Low} and T_{Pulse} (See Figure 6) are measured via software timer uptime (see above), which is passed as parameter `currentTime` to `sampleSignalDCF77()`, 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.

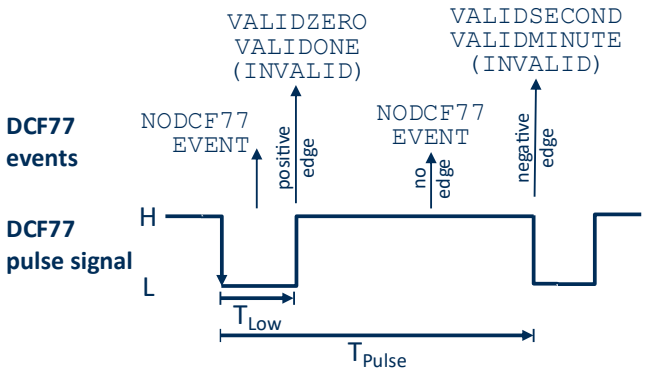

Figure 6 DCF77 events

- The function `sampleSignalDCF77()` must detect the positive and negative edges, measure times T_{Low} and T_{Pulse} and convert them into "events" (see Figure 6) stored in global variables `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 signal 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} = 100 ms ± 30ms
VALIDONE	Positive	Valid 1 bit, if T_{Low} = 200 ms ± 30ms
INVALID	Negative Positive	If T_{Pulse} is outside of the tolerance window. If T_{Low} is outside of the tolerance window.

- Whenever a DCF77 event has occurred, i.e. when the variable `dcf77Event` has been set, the "operating system" will call function `processDCF77Events()`. This 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 `processDCF77Events()` as Finite State Machine. Use the DCF77 bit number as the state and react on the events.
- When `processDCF77Events()` returns, the global variable `dcf77Event` is reset by the "operating system".

Program Design and Implementation

Task 4.1:

The architecture presented here may be considered as a proposal, only. You may come up with a different solution, as long as it fulfills the requirements.
Design your software architecture and describe it graphically with a module plan and program flow charts or using UML-style component and activity charts. The files `LED.c`, `LCD.c` and `Ticker.c`

need not be documented. Your source code must include interface descriptions for all functions and a reasonable amount of comments. For documentation requirements, see Lab 2 (Appendix A).
You may use and modify template lab3-Funkuhr-Vorlage-OS.mcp, which includes the DCF77 simulation and comes with C implementations of the LCD and the LED driver.
You may implement **your program in C** with inline assembler or **assembler subroutines** if needed.
You may use all available ANSI C library functions, for details about the C library, e.g. CodeWarrior's C compiler documentation.

When you would run the program on a real Dragon12 board, decoding the DCF77 may take up to a minute to detect a valid start and another full minute till you the complete time/date information. The simulator may run a bit faster or slower than the real Dragon12 board. Thus, developing the program with a trial-and-error approach is not really a good idea and may cause days of bug hunting. You need to carefully design your program and then incrementally test all functions individually, before you integrate them.

When your program seems to work, test the following situations:

- Does your program correctly handle different times and days, e.g. overflow at midnight? Reset the CPU and restart your program, then press and hold one of the buttons on port H.7, H.6 or H.5 to change the simulated date/time combination. Let your program run for at least 5 simulated minutes.
- Does your program correctly detect a drop-out or other faults in the DCF77 signal and resynchronize correctly when the fault disappears?
  When you press one of the buttons on port H.0 or H.1, function `readPortSim()` will generate a constant or a random DCF77 signal. The LEDs on port B.3 and B2. (see [Req 1.5]) should indicate the problem. The free running clock must continue to work normally and the LDC must never show incorrect values. If you release the button again, your program must detect a correct signal and the error LEDs must turn off.

Task 4.2: Weekdays Display, Different Time Zones

Add the following features to the radio-controlled clock:

- [Req 2.1] Display the weekday (Sun(day), Mon(day), …) in front of the date.
- [Req 2.2] Whenever button on port H.3 is pressed, the time display shall toggle between Middle European Time (=the DCF77time) and US eastern time zone, which is (DCF77time - 6 hrs). The time zone shall be indicated by “DE” or “US” in the LCD display.

Update your design (module plan, flow charts, …), implement and test your software.

Then please upload your CodeWarrior project for lab task 4.2 as ZIP-file onto Moodle including the program documentation as PDF.