Exercise 6: DCF77 Time Signal

Overview

The time signal transmitter DCF77 is a long wave sender located in Mainflingen, Germany (near Frankfurt am Main). It is used for distributing a signal that is commonly used to automatically set clocks and watches to display the correct time and date. The distributed time is actually the officially valid time in Germany. The sender has a working frequency of 77.5 kHz and a power rating of 50 kW.

In this lab course exercise, we will try to decode the DCF77 time signal using a microcontroller and display the current date and time in our debug console. For this purpose, we use a special DCF77 receiver which already provides us with a "clean" digital signal that we can directly input into the microcontroller. Since the signal quality may vary depending on the weather conditions, we have also prepared a DCF77 signal simulator that transmits a well-known date and time value. We will use that setup in case the signal quality becomes too bad.

DCF77 Time Signal Structure

The signal amplitude of DCF77 is modulated in intervals of 1 second: At the beginning of every second (except for the last second of each minute), the signal becomes low for either 100 ms or 200 ms (see also figure 1[1]). The different lengths of the low signal interval are used to binary encode information like the current minute, hour as well as the date:

- A low signal for 100 ms corresponds to a binary *zero* while a low signal for 200 ms corresponds to a binary *one*.
- The fact that the last (59th) second of a minute does not show a low signal can be used to detect the beginning of a new minute (i.e., the next time the signal is low a new minute has just started).
- The transmitted binary information correspond to the minute, hour, day, day of the week, month and year (only the last two digits) of the *upcoming* minute. Numbers are encoded as BCD (Binary Coded Digits), which means that unlike normal binary encoding, where the ith bit has weight 2^i ($i \geq 0$), the weights of the bits are: 1, 2, 4, 8, 10, 20, 40, 80, ...

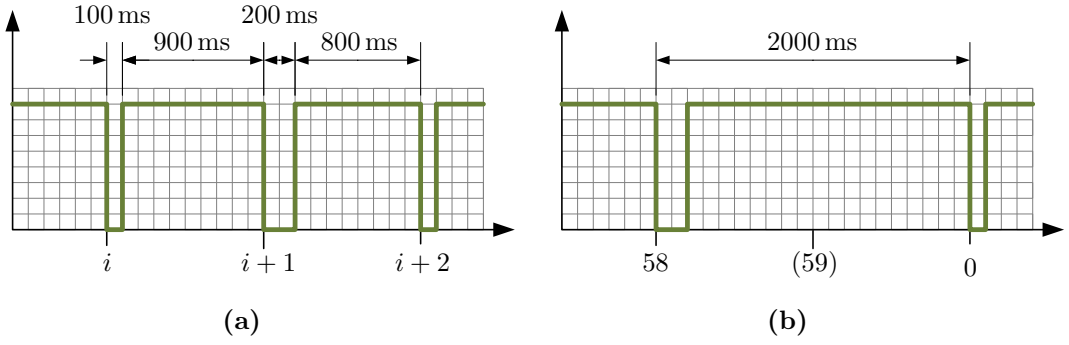

Figure 1: DFC77 Signal Plot: **(a)** Second mark: binary *zero* on the left side (100 ms low followed by 900 ms high) and binary *one* in the middle (200 ms low followed by 800 ms high); **(b)** Minute mark: there is no low signal in the 59th second.

[1]The signal shown in this figure is the one output by the receiver we use. The original terrestrial DCF77 signal actually looks somewhat different, but in this exercise we will only process the digital signal shown here.

Exercise 6.1

a) How are the following decimal numbers represented in binary and BCD notation: 7, 12, 16, 59?

b) Besides the synchronization marks for the beginning of a second and a minute, the DCF77 signal has 59 bits data payload per minute. Find out what the different bit values are used for (e.g., by "Googling").

c) Why are there error correction (parity) bits in the signal? What kind of parity is used (odd or even)?

d) Which information and/or bits (including parity bits) do we need to process if we want to obtain the following values: second s, minute n, hour h, day of the month d, day of the week w, month m and year y?

e) Imagine we have stored all data bits in a vector (array) $\vec{b} = \{b_0, b_1, \ldots, b_{59}\}$. How are the actual values for n, h, d, w, m and y calculated? Write down the formulas. What day of the week does a value of $w = 1$ correspond to?

f) The parity bits can be used to check the integrity of the received data stream. Write down formulas involving the parity bits that must hold for the signal to be considered valid.

g) Can we be sure that the received data are valid if the parity information is correct? Give an example.

h) Imagine the formula involving parity bit b_{35} does not hold. Can we still use data in $\vec{b}$?

i) What checks besides the parity check can we implement to ensure that only valid data are decoded? Take all available information into account.

j) Can you imagine why binary coded digits are used instead of binary coded numbers?

Microcontroller-based Analysis of the DCF77 Signal

Now that we know the characteristics of the DCF77 time signal, we can start writing a program for the ATmega8515 microcontroller to analyze it. For writing the program, we will need the following items from the previous lab course exercises:

- Digital I/O
- Timer with interrupts
- External interrupt or input capture unit
- UART (debugging interface)

Exercise 6.2

a) Connect the ground cable (brown) to GND on the board and the DCF77 signal cable (yellow) to an arbitrary digital input port of the microcontroller. Write a program that directly outputs the signal received at that pin on LED0. Verify that you can see the different low signal intervals and the minute mark.

b) Now collects the data bits of the DCF77 signal and detect the minute mark. Some ideas you can use in your implementation:

- When the signal quality is very bad, edge detection using input capture or external interrupt might become unreliable. To solve this problem, you might want to implement

a timer to sample the signal at a constant frequency (e.g., every 10ms) and evaluate the relative high and low time of the signal. In this case you can set up a threshold to determine if a logical zero or one is transfered in the signal.

- Remember that variables used by interrupt routines as well as the main program must be declared `volatile`.

Write debugging information to the debug console (e.g., the values of the received bits). Once all 59 bits are available and a minute mark has been seen, perform parity checks and other checks as specified in exercise 6.1 **i)** and give feedback on the debug console.

c) Once a valid data set has been obtained, format the data (e.g., `Thursday, 03.12.2009 16:30:00`) and output all available information on the debug console *each second*.

Exercise 6.3

a) Implement a button that will, when pressed, trigger another synchronization while the date and time is already known and being output to the debug console. The program should continue to output the date and time to the debug console and switch to the new date and time when it has been re-detected. This is the common behavior of DCF77-based clocks, which will typically synchronize to the time signal regularly during the night.

b) Optional: Try to make your implementation more robust by reusing information from the previously seen minute in the next minute in case the data stream from the previous minute is partially corrupted. For example, if only the minute parity was incorrect in the last round and in the next round we decode the minute correctly, we can immediatly show the correct time (note however that if the new minute is 0, which means that the previous minute was 59, we have to increment the hour and probably also the other fields).