LuxSenz receiver software

The LuxSenz receiver software runs on the STM32L031K6 microcontroller that is present on the receiver board. It processes the sampled signal data to decode received LuxSenz messages and show them on the e-ink paper.

# Building the software

The software was developed using Visual Studio and the VisualGDB plugin. Another IDE could be used too, as long as the used compiler (GCC) is the same.

# Loading the software on the microcontroller

Disconnect the power inputs from the PCB. Connect pins 3V3, SWCK, G(ND), SWIO to the ST-Link V2 USB programmer. Use VisualGDB or STLINK utility to load the program on the microcontroller.

# Display output

On the display, received sensor values and messages will be displayed large.

In the bar on the right side the following indicators can be shown:

* C Connection active
* m A message is being received
* b A bit error occurred in decoding the synchronization signal
* Z Zero-value obtained from ADC (ADC might be powered off or not connected properly)
* S Saturated value from ADC received (signal amplitude might be too high)
* F Buffer is used for more than 50%, code might be running too slow. Display timing measurements too check what part of the code is taking long
* A Previous ADC sample wasn’t saved before requesting a new ADC sample. Decrease the sample frequency, or decrease the execution time of the SysTick\_Handler.
* I Interrupt service routine of the SPI is taking too long. Usually when this occurs, the screen won’t be updated at all

On the top bar the following information can be shown:

* s: Amplitude of the digitally filtered signal, followed by the amplitude of the signal from the ADC
* q: two numbers
* t: Timing measurements: highest time measurement between successive calls to DecodeBuffer in the main loop. First number indicates the number of samples between the calls, second number points to the call number as provided in the in-code comments at every call to DecodeBuffer.
* 4/5 numbers: numbers

Instead of the received sensor values, a graph can be shown that provides information about the pulse durations as determined in the decoding algorithm. The threshold is shown in the left of the graph. When the colored pixels group to two different levels, the signal quality is good. Pixels above the line (on the left) map to pulse durations lower than the threshold (higher frequencies) and pixels below the line map to pulse durations higher than the threshold (lower frequencies). **Check this.**

On the bottom of the screen, the signal trace can be displayed instead of received messages. One can select between the unfiltered signal, the digitally filtered signal or both.

# Source files

|  |  |  |
| --- | --- | --- |
| All files in the HAL\ folders. | | Hardware Abstraction Layer libraries for the STM32 microcontroller |
| Stm32l0xx\_hal\_conf.h | | Header file to enable/disable certain modules of the HAL library |
| Startup\_stm32l031xx.c  System\_stm32l0xx.c | | Default files to let the STM32L031 start properly, made by the MCD Application Team |
| Epd2in9.h  Epdif.h  Epdpaint.h | Epd2in9.c  Epdif.c  Epdpaint.c | Source files to control the EPD (E-Paper Display), modified versions of those provided by Waveshare. |
| Fonts.h  Font8.c  Font12.c | Font16.c  Font20.c  Font24.c | Font data, used by the EPD files to draw characters on the screen. Modified versions of those provided by Waveshare. |
| Imagedata.c | | Data of images containing logos of AMS, LuxSenz and TU Delft. |
| Stm32l0xx\_it.h | | Header file of all interrupt service routines |
| Main.h | | Header file of the main source file. In this file one can:   * Select frequency settings of the physical layer of LuxSenz * Set interpolation settings to compute timestamps of threshold crossings of the physical signal * Set a Notch-filter to reject a single frequency from the physical signal (used to remove noise from LED lights) * Set default settings for showing FFT/traces of the signal on the screen * Set the used GPIO Pin numbers |
| Stm32l0xx\_it.c | | Empty interrupt handlers for errors and USART (if USART is enabled) |
| Stm32l0\_hal\_msp.c | | GPIO Initialization code that is called by the HAL library |
| Initializations.c | | GPIO and Clock initialization code, not handled by the HAL library |
| Main.c | | All the important stuff |

# Code structure

This section describes the code structure of the main program, completely implemented in the file main.c. The flow of the receiver software is shown schematically on the next page, describing how samples are processed in real-time.

In Main.c two interrupt service routines (ISR) are defined. One ISR is the systick handler, the other is the SPI interrupt handler. All other pieces of code are called by the main program loop.

## Interrupt service routines

The first ISR is the systick handler. The systick timer is programmed to call the systick handler 10.000 times per second. There is some tolerance in this value: measured was approximately 10.020 calls per second, resulting in the same number of samples per second.

The systick handler activates the SPI peripheral. By activating the SPI, a sample is requested from the ADC and (buffered) data can be sent to the e-ink display. The SPI interrupt handler is called when the SPI peripheral has received the sample from the ADC.

## Program flow chart

The flow chart below describes the real-time flow of ADC sample data towards displaying received messages.

## Use of SPI peripheral

The way the SPI peripheral is used, is quite complex. The SPI1 peripheral is used to simultaneously transmit 8 bits of data to the e-ink display and receive 16 bits of data from the ADC. It was possible to implement the SPI communications, because the ADC does not need to receive data, the e-ink display doesn’t transmit data, and both can run on the same clock signal.

The SPI1 is programmed in 16-bit bidirectional mode, thus it will transmit and receive 16 bits of data very time it is activated. When SPI1 is activated, the CS (clock signal) to the ADC is pulled low to activate SPI communication with the ADC. The SPI1 receives 16 bits from the ADC, containing the 12-bit sample (zero-padded to 16 bits). The CS to the e-ink display is controlled with TIM2, a hardware timer that is reset each time when the SPI1 is activated. When there is data available to transmit to the e-ink paper, the value of TxBufferStatus is non-zero. Then, the TIM2 is set in the systick handler, such that the CS to the e-ink display is pulled low after transmitting the first 8 bits (that do not contain the data, but are equal to zero). This way, the e-ink only receives the last 8 bits that contain the data. The CS will be pulled high again when SPI1 has finished.

The e-ink display distinguishes between receiving data and receiving commands over SPI, and therefore requires the DC (data/command) control signal. When SPI1 is activated, the DC signal is set accordingly: the value of TxBufferStatus will be different for data or commands.

## USART

Some parts of code are disabled, that otherwise could help to enable UART/USART communication with a PC. It could be used to implement saving data from a sampled signal trace.

## Potentiometers

Originally, the LuxSenz receiver was meant to have two digital potentiometers to make the hardware filters and amplifiers tuneable. The behaviour of these potentiometers was non-ideal, causing the amplification circuit to saturate. By defining USE\_POT in main.h parts of code can be enabled that were meant to control the potentiometers. Currently, those parts of code are not used. Control of the potentiometers was implemented using the U/D-protocol of the MCP4012.

# Correcting the sample frequency

In the file main.h two macro’s are defined: SAMPLE\_FREQ and SYSTICK\_FREQ. SAMPLE\_FREQ matches the desired sample frequency (10000 samples per second) and should not just be changed. Design of filter coefficients is always related to the designed sample frequency. SYSTICK\_FREQ is used to setup the system timer. In an ideal case, the timer interrupt frequency matches the number set to SYSTICK\_FREQ, then the value of SYSTICK\_FREQ should be equal to SAMPLE\_FREQ. Because of small errors in the oscillator frequencies, the actual interrupt frequency will be slightly off. The value of SYSTICK\_FREQ should be adjusted to compensate these errors, such that the actual interrupt frequency (sample frequency) match the value of SAMPLE\_FREQ. The sample frequency can be measured by connecting an oscilloscope to resistor R2 on the PCB (ADC Enable signal).

# Troubleshooting

* **Q: The e-ink display continuously toggles between 100% white and a normal screen**
* A: After sending data to the e-ink display, the program doesn’t wait long enough for the display to process this data. Search for calls to EPD\_SetMemoryPointerDontWait and EPD\_DisplayFrameDontWait and increase the number of iterations of the loops following these calls.
* **Q: The receiver always shows the saturation indicator, even if I block the incoming light.**
* A1: It can be that one of the batteries is empty (voltage 3.3V or lower). In this case, the indicated amplitude of the unfiltered signal is usually higher than the indicated amplitude of the filtered signal. Charging the battery will solve the issue.
* A2: The amplification circuits might amplify the signal too much, such that it saturates. Blocking a part of the incoming light might help, as well as adjusting the amplification circuit.
* **Q: The e-ink screen becomes largely grey, instead of black with white**
* A: one of the batteries is empty. Recharge the batteries.
* **Q: Everything on the e-ink screen is displayed on the wrong position**
* A: Ocassionally it can occur that the e-ink screen writes/reads on the wrong memory locations, especially after issues with poorly connected wires. Disconnect power, wait a few minutes and reconnect the power.